Markt&Technik

Im Buch enthalten:
unbestückte Experimentier-Platine

Richard Schmidt
Dr. Peter Wratil

Atari ST

MESSEN
STEUERN
REGELN

Angewandte Interface-Technik

Aufbau ★ Funktion ★ Schaltungslogik
★ Schnittstellen-Parameter

Auf Diskette enthalten:
alle im Buch vorgestellten Steuerprogramme in Assembler und Basic

Atari ST
Messen · Steuern · Regeln

Richard Schmidt · Dr. Peter Wratil

Atari ST

Messen · Steuern · Regeln

Angewandte Interface-Technik

Aufbau · Funktion · Schaltungslogik
Schnittstellen-Parameter

Markt&Technik Verlag AG

CIP-Titelaufnahme der Deutschen Bibliothek

Schmidt, Richard:
Atari ST – messen, steuern, regeln :
angewandte Interface-Technik ;
Aufbau, Funktion, Schaltungslogik, Schnittstellen-Parameter /
Richard Schmidt ; Peter Wratil. –
Haar bei München : Markt-u.-Technik-Verl., 1989
ISBN 3-89090-679-6
NE: Wratil, Peter:

Die Informationen in diesem Produkt werden ohne Rücksicht auf einen eventuellen Patentschutz veröffentlicht.
Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt.
Bei der Zusammenstellung von Texten und Abbildungen wurde mit größter Sorgfalt vorgegangen.
Trotzdem können Fehler nicht vollständig ausgeschlossen werden.
Verlag, Herausgeber und Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.
Für Verbesserungsvorschläge und Hinweise auf Fehler sind Verlag und Herausgeber dankbar.

Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien.
Die gewerbliche Nutzung der in diesem Produkt gezeigten Modelle und Arbeiten ist nicht zulässig.

Die Platine wurde vom Verlag bestückt und mit der veröffentlichten Software intensiv getestet. Soft- und Hardwarefehler konnten nicht festgestellt werden. Trotzdem haftet der Markt&Technik-Verlag nicht für Schäden jeglicher Art, die auf den Einbau zurückzuführen sind. Außerdem: Bedenken Sie, daß durch das Öffnen des Gerätes die Garantie erlischt.
Vorsicht, der Computer wird mit 220 Volt betrieben.
Bei geöffnetem Gerät stets die Stromzufuhr unterbrechen.

Atari ST ist ein eingetragenes Warenzeichen der Atari Corp., USA

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

92 91 90 89

ISBN 3-89090-679-6

© 1989 by Markt&Technik Verlag Aktiengesellschaft,
Hans-Pinsel-Straße 2, D-8013 Haar bei München/West-Germany
Alle Rechte vorbehalten
Einbandgestaltung: Grafikdesign Heinz Rauner
Dieses Produkt wurde mit Desktop-Publishing-Programmen erstellt
und auf der Linotronic 300 belichtet
Druck: Kösel, Kempten
Printed in Germany

Inhaltsverzeichnis

Vorwort 9

Einleitung und Überblick 11

1 **Grundzüge der Logikentwicklung** 19

1.1 Logische Verknüpfungen, binäre Zustände 19
1.2 Wahrheitstafeln, Symbole 24
1.3 Vereinfachungs- und Umformungsmethoden 25
1.4 Die Methode von Karnaugh und Veitch 27
1.5 Logische Schaltzeichen, Logikfamilien 29
1.6 Sequentielle logische Schaltungen 32

2 **Aufbau und Funktionsweise von Prozessoren und Systemen** 37

2.1 Geschichtlicher Überblick 37
2.2 Struktur von Zentralprozessoren 38
2.3 Die Hardware des Prozessors 68000 45
2.4 Kundenspezifische Integration 47

3 **Datentransport** 51

3.1 Kommunikationswege und Datentransportstrukturen 51
3.2 Transportmedien 53
3.3 Parallele Bussysteme innerhalb des Rechners 53
3.4 Parallele Bussysteme außerhalb des Rechners 57
3.5 Buszugriffsverfahren 62
3.6 Parallele sternförmige Verbindungen (Beispiel Centronics) 65
3.7 Serielle Verbindungen 67
3.8 Normungen beim Datentransport 69
3.9 Datensicherung 71

4 **Mikroprozessorbausteine** 73

4.1 Allgemeine Betrachtungen 73
4.2 Die Zentraleinheit MC 68000 74
4.3 Der Multifunktionsbaustein MPF 68901 80
4.4 Der Disketten-Controller WD 1772 87
4.5 Der serielle Baustein ACIA 6850 89
4.6 Der Sound-Prozessor YM 2149 90
4.7 Der Parallelbaustein 8255 oder 82C55 91
4.8 Der serielle Baustein 8251 98
4.9 Speicherbausteine 103

5 **Aufbau und Funktionsweise des Atari ST** 105

5.1 Hardwareüberblick 105
5.2 Der Input-/Output-Bereich 110
5.3 Direct Memory Access – DMA 113
5.4 Die Interruptstruktur 114
5.5 Weitere Schnittstellen 116

6 **Verarbeitung analoger Daten** 121

6.1 Analoge Signale und analoge Pegel 121
6.2 Der Operationsverstärker 122
6.3 Einfache Anwendungen von Operationsverstärkern 124
6.4 Verfahren zur Analog-Wandlung digitaler Werte 130
6.5 Analog-/Digital-Wandlung durch das Rampenverfahren 133
6.6 Doppelrampenverfahren zur Analog-/Digital-Wandlung 135
6.7 Die Methode der sukzessiven Approximation 136
6.8 Spezielle Konversionsverfahren 138
6.9 Fehler der Konversionsverfahren 141

7 **Programmieren mit dem Personalcomputer** 145

7.1 Programmiersprachen im Überblick 145
7.2 Schritte zur Programmentwicklung 147
7.3 Programmiermodelle des 68000 150
7.4 Adressierungsarten beim 68000 156
7.4.1 Direkte Registeradressierung 157
7.4.2 Absolute Adressierung 157
7.4.3 Unmittelbare oder Konstanten-Adressierung 158

7.4.4 Adreßregister indirekt (ARI) 158
7.4.5 Adreßregister indirekt mit Postinkrement 159
7.4.6 Adreßregister indirekt mit Predekrement 160
7.4.7 Adreßregister indirekt mit Adreßdistanz 160
7.4.8 Adreßregister indirekt mit Adreßdistanz und Index 161
7.4.9 PC-relative Adressierung 162
7.4.10 PC-relative Adressierung mit Index 162
7.5 Die Assemblersprache des 68000 im Überblick 163
7.5.1 Datentransferbefehle 165
7.5.2 Logische und arithmetische Befehle 166
7.5.3 Befehle zur Verwaltung der Zentraleinheit 168
7.5.4 Sprungbefehle 169
7.5.5 Interruptbefehle 171

8

Aufbau und Anwendung der Interface-Karte 173

8.1 Bestückung und Test der Interface-Karte 173
8.2 Funktionsweise der Interface-Platine 181
8.3 Einfache Anwendungen der Platine 185
8.4 Anwendungen und Programmierung in Assembler und Pascal 190
8.5 Betrieb eines Logikanalysators mit der Interfacekarte 194
8.6 Druckerinterface für parallele Drucker 204
8.7 Weiterer Ausbau der Interface-Karte 207
8.8 Analoge Signalausgabe 209
8.9 Regelbare Spannungsquelle 212
8.10 Regelbare Stromquelle für geerdete Verbraucher 215
8.11 Interface für analoge Eingabe 217
8.12 Interface für ein ADC nach dem Rampenverfahren 217
8.13 Interface nach der Methode der sukzessiven Approximation 221
8.14 Anschluß von Sensoren am Beispiel der Temperaturmessung 223
8.15 Speicheroszilloskop für analoge Signale 226
8.16 Zweipunktregler für Temperaturstabilisierung 231
8.17 Inbetriebnahme des seriellen Interfaces 240
8.18 Serielle Dateneingabe 244
8.19 Serielle Ausgabe und Terminal-Betrieb 247
8.20 Einstieg in lokale Netze 251

9 **Anhang** 257

9.1 Fehlerbehebung auf der Interface-Karte 257
9.2 Farbcode-Tabelle 259
9.3 Der Befehlssatz des Prozessors MC 68000 260
9.4 ASCII-Tabelle 263
9.5 Symbole zur Erstellung von Flußdiagrammen 264
9.6 Bezugsquelle für weitere Interface-Karten 265
9.7 Literaturverzeichnis 266

Stichwortverzeichnis 271

Hinweise auf weitere Markt & Technik-Produkte 277

Vorwort

Der Atari ST erfreut sich mittlerweile einer großen Beliebtheit, was auch in der unüberschaubaren Zahl von Büchern, Zeitschrifen und Programmen für diesen Computer zum Ausdruck kommt.

Die weite Verbreitung des Atari liegt in seiner hohen Leistungsfähigkeit, verbunden mit einem geringen Preis, begründet. Auch das Softwareangebot kann professionellen Ansprüchen durchaus genügen. Hier bestechen die komfortabe Benutzeroberfläche und die guten grafischen Möglichkeiten.

Dagegen ist das Angebot an speziellen Hardware-Erweiterungen eher bescheiden. Das liegt zum Teil sicher an der fehlenden Buserweiterung, die den Anschluß nicht vorgesehener Hardware sehr erschwert (im Gegensatz z.B. zum PC oder zum schon fast legendären Apple IIe). Das ist um so bedauerlicher, da der Atari durch seine Hochleistungs-CPU auch in diesem Bereich sehr gut eingesetzt werden könnte.

Das vorliegende Buch soll helfen, diese »Hardware-Lücke« beim Atari zu schließen. Die mitgelieferte Platine ermöglicht den Aufbau eigener Schaltungen für die unterschiedlichsten Anwendungen.

Hierbei haben wir nicht nur den engagierten Bastler im Blick, sondern auch professionelle Anwendungen in Forschung und Lehre. Gerade an Schulen, Universitäten und Instituten ist der Atari weit verbreitet. Hier kann die mitgelieferte Platine z.B. zum Erfassen von Meßdaten verwendet werden, die auf dem Atari dann auch dargestellt und ausgewertet werden.

Ziel dieses Buches ist es aber auch, den Leser in die Technik der Interface-Schaltungen einzuführen, womit Probleme aus dem Bereich Messen, Steuern, Regeln gelöst werden können. Anhand von vielen Beispielen aus diesem Bereich werden wichtige Grundlagen erarbeitet und praktisch vorgeführt.

Wir möchten an dieser Stelle allen danken, die zum Entstehen dieses Buches beigetragen haben.

Besonders danken möchten wir unseren Frauen, Christa-Carola und Marina, für ihre Geduld und ihr Verständnis, ohne die dieses Buch sicher nicht entstanden wäre.

Köln

Richard Schmidt
Dr. Peter Wratil

Einleitung und Überblick

Dieses Buch wendet sich nicht nur an Computerprofis, Hacker und Elektronikbastler, sondern soll auch diejenigen erreichen, die das Innenleben ihres Atari ST verstehen wollen, ohne bisher über die notwendigen Vorkenntnisse zu verfügen. In diesem Buch wurde versucht, alle für das Arbeiten mit der Interface-Karte notwendigen Kenntnisse zu vermitteln. Deshalb erläutert diese Einleitung die Funktionsweise eines Computers für Anfänger. Fortgeschrittene können sie daher getrost überspringen und direkt mit dem ersten Kapitel beginnen (es sei denn sie wollen sich einen kurzen Überblick über den Inhalt des Buchs verschaffen, der am Ende dieser Einleitung gegeben wird).

Zunächst noch ein Wort zu den im Buch verwendeten Begriffen: Da die Computertechnik vorwiegend im englischsprachigen Raum und hier vor allem in den USA entwickelt wurde, kommen fast alle Fachwörter aus der englischen Sprache. In den deutschen Fachbüchern oder Bedienungsanleitungen sind diese häufig übersetzt, wodurch manchmal Mißverständnisse entstehen. In einigen Fällen weiß der Leser nicht einmal mehr, was mit dem deutschen Begriff gemeint ist, wenn er die englische Bezeichnung kennt (die aber nicht dabei steht). Gerade die Handbücher von Big Blue (Kosename für den größten Computerkonzern der Welt) sind hierfür ein abschreckendes Beispiel, doch zum Glück hat der Benutzer des Atari ST zumindest damit nichts zu tun. Aber bitte keine Schadenfreude – manche Ataribücher sind auch nicht viel besser (natürlich nur die von der Konkurrenz). In diesem Buch wurden Fachwörter überall dort verwendet, wo es sich um feste Begriffe handelt, die sich als Teil der EDV-Fachsprache eingebürgert haben, ansonsten werden deutsche Bezeichnungen benutzt (für machen Leser mögen es noch viel zu viele englische Begriffe sein, aber daran läßt sich jetzt nichts mehr ändern – dieses Computer-Chinesisch ist einfach Teil einer Fachsprache). Die englischen Begriffe sind jeweils beim ersten Vorkommen in ihrer Bedeutung übersetzt oder erläutert.

Jeder funktionierende Computer besteht aus Hard- und Software. Die Hardware ist das, was man anfassen und notfalls mit Lötkolben und Zange reparieren kann. Die Software ist viel schwerer zu fassen. Es ist die Gesamtheit der Programme für einen Computer. Man könnte poetisch sagen: Wenn die Hardware der Körper eines Computers ist, so wird ihm durch die Software die Intelligenz eingegeben. Profaner gesagt: Ein Computer ohne Software ist ein Haufen Schrott, mit dem man nichts anfangen kann. Erst die Software macht daraus eine vielseitig einsetzbare Maschine zum Lösen der verschiedensten Aufgaben. Wenn man z.B. eine Taste betätigt und das entsprechende Zeichen erscheint auf dem Bildschirm, so wird zur

Bewältigung dieser Aufgabe ein kompliziertes Zusammenspiel von Hard- und Software benötigt.

Hard- und Software bilden für den Anwender eine Einheit, und es ist für den Anfänger – und manchmal auch für den Fachmann – nicht ganz einfach zu durchschauen, wie diese beiden Komponenten eines Computers zusammenwirken. Diese Probleme werden beim Aufbau und Programmieren der beiliegenden Karte noch näher erläutert. Um die Funktionsweise eines Computers zu verstehen, muß man bei Aufbau und Funktion der Hardware beginnen. Hierbei kann man verschiedene Funktionsgruppen unterscheiden, die in Bild 0.1 dargestellt sind:

- der Systembus
- die Zentraleinheit (CPU, Central Processing Unit)
- der Speicher für Programm und Daten
- die Anschlüsse für die Peripherie wie Tastatur, Monitor, Drucker u.ä.

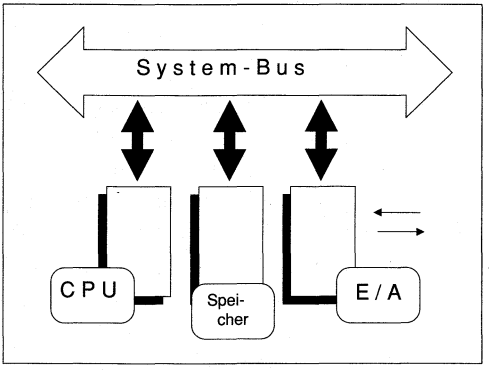


Bild 0.1: Prinzipieller Aufbau der Hardware eines Computers

Die Funktionsgruppen eines Computers müssen Daten austauschen, was über den sogenannten Systembus geschieht. Dieser Bus besteht aus einer Anzahl von Leitungen, auf denen die Daten transportiert werden. Über diese Leitungen sind alle Komponenten eines Computers miteinander verbunden. Der Begriff Bus ist hierbei durchaus wörtlich als das zu nehmen, was er auch sonst bedeutet: ein Transportmittel. Transportiert werden hierbei Informationen. Die einzelnen Baugruppen eines Computers kann man sich dabei als Haltestellen vorstellen, an denen die Informationen ein- und aussteigen.

Da beim Atari ST alle Komponenten auf einer Platine untergebracht sind, ist der hier beschriebene Aufbau nicht so deutlich sichtbar wie bei anderen Computermodellen, wo der Bus z.B. durch eine Anzahl paralleler Leitungen aufgebaut ist. Dieses Konzept ist jedoch allgemeingültig und beim Atari ST genauso realisiert wie bei allen anderen Rechnern basierend auf dem Von-Neumann-Prinzip (dazu später mehr).

Zunächst zu den Daten: Ein Computer ist im Prinzip eine Maschine, die aus einer unvorstellbar großen Anzahl von Schaltern besteht, die durch ein Programm sehr schnell betätigt werden können. Ein Schalter kann sich in zwei Zuständen befinden: er ist entweder offen oder geschlossen. In einem Stromkreis bedeutet dies, daß bei offenem Schalter kein Strom fließt, bei geschlossenem fließt Strom. Diese beiden Zustände nennt man High oder 1 (Strom) und Low oder 0 (kein Strom). In einem Computer stellen diese beiden Zustände die kleinste mögliche Informationseinheit dar, die als 1 Bit (Abkürzung für Binary Digit = Binärzahl) bezeichnet wird. Binäre Zustände lassen sich als Zahlen eines Binärsystems auffassen, mit denen man genauso rechnen kann wie mit den bekannten Dezimalzahlen, was in Kapitel 1 ausführlich beschrieben wird. Alle Informationen in einem Computer, ob es sich um Zahlen, Buchstaben oder sonst etwas handelt, sind aus solchen Binärzahlen zusammengesetzt. Auch die Leitungen des Systembusses können nur diese beiden Zustände annehmen. Jede Leitung des Busses kann man sich als eine Stelle in einer Binärzahl vorstellen. Eine Kombination von 8 Bit bezeichnet man als 1 Byte.

Die Daten werden von der Zentraleinheit oder auch CPU verarbeitet, die das zentrale Steuer- und Rechenwerk des Computers ist. In dieser Einleitung kann nicht darauf eingegangen werden, wie die CPU funktioniert (siehe hierzu die Kapitel 2, 5 und 7), deshalb stellt man sich die Zentraleinheit zunächst als schwarzen Kasten vor, der bestimmte Befehle ausführt. Da über den Bus nur Binärzahlen übertragen werden können, müssen auch die Befehle an die Zentaleinheit Binärzahlen sein, die von der CPU als Anweisungen für ganz bestimmte Operationen interpretiert werden. Als Beispiel sei angenommen, die Zahl 23 ist für eine CPU der Befehl zum Addieren zweier Zahlen. Immer wenn dieser CPU der Befehl 23 geschickt wird, addiert sie also zwei Zahlen, die ihr vorher durch andere Anweisungen mitgeteilt werden müssen. Entsprechendes gilt für alle anderen Befehle.

Alle Befehle liegen im Speicher und werden über den Datenbus zur Zentraleinheit gebracht. Bei den meisten kleineren und mittleren Computern ist dieser Speicher byteweise organisiert. Das bedeutet, die kleinste Speichereinheit, die angesprochen werden kann, ist ein Byte. Dies stammt aus einer Zeit als die Zentraleinheiten des Computer Daten nur in Portionen von 8 Bit, also einem Byte, verarbeiten konnten. Man spricht dann auch von einer 8-Bit-CPU. Der Atari ST hingegen hat als Zentraleinheit eine CPU der nächst höheren Generation, die in der Lage ist, Daten in Portionen von 16 Bit zu verarbeiten, entsprechend spricht man hier von einer 16-Bit-CPU. Das Konzept, den Speicher byteweise zu verwalten, wurde aber auch bei den meisten 16-Bit-Zentraleinheiten – auch bei der des Atari-PC – beibehalten. Für das Ansprechen des Speichers gibt es zwischen einer 8- und einer 16-Bit-CPU aber keinen großen Unterschied, da ja immer eine Speicherzelle, egal ob 8 oder 16 Bit breit, angesprochen wird. Jede Speicherstelle kann man sich als eine Anzahl von Schaltern vorstellen. Bei einem 8-Bit-Speicher sind es acht Schalter, bei einem 16-Bit-Speicher eben sechzehn. Je nachdem ob die Schalter offen oder geschlossen sind, speichern sie eine 0 (offen) oder eine 1 (geschlossen). In einem Computer werden diese Schalter natürlich nicht von Hand betätigt, sondern sind als Schaltelemente in einem integrierten Baustein, auch IC (Integrated Circuit, integrierter Baustein) oder Chip genannt, untergebracht. Das Prinzip bleibt aber gleich.

Wie greift die CPU nun auf den Speicher zu? Im Prinzip funktioniert das wie bei der Anschrift oder besser noch der Telefonnummer einer Person. Will man dieser Person etwas mitteilen, so wählt man ihre Telefonnummer und erhält hoffentlich eine Verbindung. Ganz analog ist es bei den Speicherplätzen: Jeder hat eine eindeutige Adresse. Da es im Computer nichts anderes als Zahlen gibt, ist die Speicheradresse ebenfalls eine Zahl. Beim Atari ST ist dies eine vierundzwanzigstellige Binärzahl. Damit kann man von 0 bis 16.777.200 zählen. (Dieser Zahlenumfang entspricht 16 Megabyte oder 16 Mbyte.) Um diese Speicherstellen im Computer anzusprechen, sind im Systembus des Atari ST 23 Adreßleitungen reserviert. (Warum es nur 23 Adreßleitungen sind und nicht 24, wird etwas weiter unter erklärt). Man nennt dies auch den Adreßbus. Neben dem Adreß- und Datenbus gibt es noch weitere Leitungen, z.B. solche mit denen die CPU den Speicherbausteinen mitteilen kann, ob jetzt eine gültige Adresse auf den Adreßbus liegt und ob sie die Daten in den Speicher schreiben oder aus dem Speicher lesen will. Diesen Bus nennt man auch Kontrollbus. Wenn die CPU Daten von einem Speicherplatz lesen will, so muß sie also folgende Aktionen ausführen: Als erstes wird die Adresse dieses Speicherplatzes auf den Adreßbus gelegt. Kurze Zeit später wird den Speicherbausteinen durch ein entsprechendes Signal des Kontrollbusses die Gültigkeit der Adressen mitgeteilt. Außerdem wird das Signal Daten-lesen des Kontrollbusses aktiviert. Vor den Speicherbausteinen befindet sich eine Logik, die anhand der Daten auf dem Adreßbus erkennt, in welchen Speicherbausteinen sich die gewünschte Information befindet und diese dann aktiviert.

Die so angesprochenen Speicherbausteine legen die Daten der angeforderten Adresse auf den Datenbus, wo die CPU sie abholen kann. Hierbei müssen diese Speicherbausteine nur die internen Schalter (offen oder geschlossen) der angeforderten Adresse mit dem Datenbus verbinden. (In der Praxis treten dabei natürlich nicht unerhebliche Probleme auf, aber es geht hier ja nur um das Prinzip.) Tabelle 0.1 zeigt die Organisation des Speichers bei einem Computer mit 24 Adreß- und acht Datenleitungen. Hierbei entspricht jeder Leitung eine Binärstelle):

Adressen (24 Leitungen)	**Daten (acht Leitungen)**
0000 0000 0000 0000 0000 0000	0101 1001
0000 0000 0000 0000 0000 0001	0110 0011
0000 0000 0000 0000 0000 0010	1101 1101
0000 0000 0000 0000 0000 0011	0101 0111
.	.
.	.
.	.
1111 1111 1111 1111 1111 1110	0110 0110
1111 1111 1111 1111 1111 1111	1110 1101

Tabelle 0.1: Beispiel für die Organisation des Speichers mit 24 Adreß- und acht Datenleitungen

Die Organisation des Speichers im Atari ST ist etwas anders aufgebaut als in Tabelle 0.1 dargestellt. Wie bereits erwähnt, ist der Speicher auch hier byteweise organisiert (jedes Byte hat eine eindeutige Adresse), aber da der Datenbus der CPU vom Atari-PC 16 Bit (also 2 Byte) umfaßt, können bei jedem Zugriff auf den Speicher gleich 2 Byte transportiert werden. Das heißt aber anders ausgedrückt, man braucht nur jede zweite Adresse (nur die geraden 0, 2, 4, ...), um alle Informationen aus dem Speicher zu verwalten. Deshalb fehlt beim Atari ST die Adreßleitung A0 und die Speicherorganisation sieht aus wie in Tabelle 0.2.

Adressen (23 Leitungen)	**Daten (16 Leitungen)**
0000 0000 0000 0000 0000 000	0101 1001 0110 0011
0000 0000 0000 0000 0000 001	1101 1101 0101 0111
.	.
.	.
.	.
1111 1111 1111 1111 1111 111	0110 0110 1110 1101

Tabelle 0.2: Organisation des Speichers beim Atari ST

Neben den Befehlen sind im Speicher auch Daten untergebracht. Bei dem Addierbeispiel sollten z.B. zwei Zahlen addiert werden. Doch wo kommen diese Zahlen überhaupt her? Neben den Befehlen gibt es im Speicher auch Daten, die nichts anderes als Zahlen im herkömmlichen Sinne sind, z.B. die beiden Zahlen, die durch den Befehl 23 addiert werden sollen. Jede CPU verfügt über Befehle, die diese Daten aus dem Speicher holen. Hier soll angenommen werden, die Zahl 22 sei der Befehl zum Holen eines Datenbytes aus dem Speicher. Und zwar soll das Datenbyte dabei als Konstante direkt hinter dem Befehl 22 stehen. Ein solcher Befehl besteht also aus einem Code-Teil, den man auch Opcode (von engl. Operation Code – Befehlscode) nennt, und einem Datenteil, oft auch Operand genannt. Bei der CPU des Atari ST besteht jeder Opcode aus einem Wort. Die CPU holt bei jedem Befehl zunächst das erste Byte oder Wort und erkennt daran, ob für diesen Befehl noch weitere Bytes oder Worte benötigt werden. Sollte dies der Fall sein, so werden auch diese aus dem Speicher geholt und der Befehl ausgeführt.

Ein Computerprogramm für die Zentraleinheit besteht also aus einer Folge von Zahlen im Speicher. Tabelle 0.3 zeigt ein einfaches Addierprogramm für eine fiktive 8-Bit-CPU (der Opcode für jeden Befehl soll hier jeweils ein Byte lang sein). In der ersten Spalte stehen die Adressen, in der nächsten die Informationen, die auf diesen Adressen gespeichert sind, danach folgt eine Erläuterung zu den Befehlen im Klartext. Mit einer solchen Erläuterung kann eine Zentraleinheit nichts anfangen, für sie gibt es nur Adressen und Daten. In diesem Beispiel beginnt das Programm bei der Adresse 0.

Mit dem ersten beiden Befehlen (Code 22) wird eine Zahl in die Zentraleinheit geladen. Diese Befehle bestehen aus zwei Byte, ein Byte Opcode und ein Byte Operand. Der dritte Befehl (Code 23) addiert diese beiden Zahlen und besteht nur aus einem Opcode, da er zur Ausführung keine Daten mehr benötigt. Der letzte Befehl (Code 24) schließlich legt das Ergebnis wieder im Speicher ab und benötigt hierfür natürlich die Adresse, wo der Wert abgespeichert werden soll. Diese Adresse folgt als Operand im nächsten Byte, so daß auch dieser Befehl aus zwei Byte besteht. Als Ergebnis dieses kleinen Programms steht auf der Adresse 107 die Summe der Addition. Die Zahl 5 steht natürlich erst dann auf der Adresse 07, wenn das Programm ausgeführt wurde.

Adresse	**Inhalt**	**Befehl im Klartext (nur für den Menschen)**
00	22	Ein Byte aus dem Speicher holen (Opcode)
01	2	Datenteil zu dem Ladebefehl (Operand)
02	22	Ein Byte aus dem Speicher holen (Opcode)
03	3	Datenteil zu dem Ladebefehl (Operand)
04	23	Zwei Zahlen addieren
05	24	Ergebnis abspeichern (Opcode)
06	107	Datenteil zum Speicherbefehl (Operand)
07	5	Ergebnis, das von der CPU berechnet und hier gespeichert wurde

Tabelle 0.3: Beispiel für ein einfaches Computerprogramm

Da sowohl die Befehle als auch die Daten im Speicher als Zahlen abgelegt sind, drängt sich die Frage auf, wie die Zentraleinheit zwischen diesen Daten und den Befehlen unterscheiden kann. Die Antwort ist denkbar einfach: Die CPU kann zwischen Daten und Befehlen nicht unterscheiden. Fast alle heute verwendeten Computer, ob es sich um einen kleinen Heimcomputer oder einen Superrechner handelt, sind nach der sogenannten Von-Neumann-Architektur (nach ihrem Begründer benannt) aufgebaut. Bei diesem Konzept wird zwischen Befehlen und Daten nicht unterschieden. Um das zu verstehen, kann man betrachten, wie der Computer ein Programm abarbeitet.

Nach dem Einschalten holt die CPU aus einer festgelegten Speicherstelle den ersten Befehl. Der Einfachheit halber soll hier angenommen werden, die CPU beginnt mit der Adresse 0. Nach dem Einschalten muß an dieser Speicherstelle der erste sinnvolle Befehl stehen. Die Zentraleinheit holt also von der Adresse 0 das erste Byte des ersten Befehls. Besteht dieser Befehl nur aus einem Byte, so kann er schon ausgeführt werden. Besteht der Befehl dagegen aus mehreren Bytes, so müssen zunächst diese von den Adressen 1, 2 usw. geholt werden, bevor der Befehl abgearbeitet werden kann. Hierbei merkt sich die CPU intern, an welcher Stelle des Speichers der nächste Befehl steht. Ist der erste Befehl vollständig ausgeführt, wird mit Hilfe des internen Befehlszeigers der zweite geholt usw. Ob ein Byte im Speicher ein Befehl oder sonst irgendetwas ist, hängt also nur von der Stelle im Programm ab. Passiert

hier ein Fehler, weil der Speicher z.B. zufällige Zahlen beinhaltet, so wird der Computer unweigerlich abstürzen, weil er kein sinnvolles Programm mehr findet. Man muß dann wieder ganz neu beginnen, in unserem Beispiel bei Adresse 0. Dies erreicht man z.B. durch Ausschalten des Computers. Das oben angeführte Beispielprogramm würde den Computer also vermutlich zum Absturz bringen, da mitten im Programm (an der Adresse 107) ein Datum abgespeichert wurde. Nach dem letzten sinnvollen Befehl (Adressen 105 bis 106) würde die CPU den nächsten Befehl von der Adresse 107 holen. Dieser Befehl ist aber gar kein Befehl, sondern eine Zahl, was die CPU aber nicht weiß. (Sie ist ja nur eine Maschine, die gemäß ihrer Bauweise funktioniert.) Sie führt an dieser Stelle einfach einen Befehl mit dem Opcode 5 aus, was immer das auch bewirken mag.

Neben der Zentraleinheit und dem Speicher benötigt ein Computer auch Ein- und Ausgabegeräte (abgekürzt E/A) wie Tastatur, Monitor, Diskettenlaufwerk, Festplatte, Drucker u.ä. Diese Geräte können nicht direkt an den Bus angeschlossen werden, hierfür sind weitere Baugruppen notwendig, da die Daten für das entsprechende Peripheriegerät erst aufbereitet werden müssen. Solche Funktionseinheiten nennt man auch Interface-Baugruppen. Um diese Baugruppen anzusprechen, gibt es verschiedene Möglichkeiten. Hier soll nur die Methode erläutert werden, die im Atari ST Verwendung findet.

Wie schon weiter oben erläutert, werden die Speicherbausteine nicht direkt durch den Adreßbus angesprochen, sondern dazwischen befindet sich eine Logik, die entscheidet, welche Speicherchips durch die aktuelle Adresse ausgewählt werden. Nun gibt es aber überhaupt keinen Grund, warum bei einer bestimmten Adresse überhaupt ein Speicherbaustein angesprochen werden muß. Vielmehr könnte die Logik auch eine ganz andere Einheit aktivieren, z.B. eine der eben erläuterten Peripherieeinheiten, wie den Drucker. Und genau so funktioniert das E/A-Konzept beim MC 68000 und damit auch beim Atari ST. Ein bestimmter Bereich des Speichers ist für Ein-/Ausgabeeinheiten reserviert und die Logik in einem der Chips erkennt bei jeder Adresse, ob hier ein Speicherbaustein oder eine E/A-Einheit angesprochen werden muß. Ausführlicher wird das hier beschriebene E/A-Konzept des MC 68000 im Kapitel 4.2 behandelt.

Die diesem Buch beiliegende Interface- oder auch Prototypenkarte arbeitet genauso. Der auf dieser Platine fest verdrahtete Teil dient dazu, bestimmte Leitungen des Adreß- und Kontollbus zu dekodieren, um festzustellen, wann die Karte von der Zentraleinheit angesprochen wird. In den folgenden Kapiteln des Buches wird auf diese Themen noch genauer eingegangen. Zum Abschluß dieser Einleitung soll noch ein kurzer Überblick über den Inhalt der einzelnen Kapitel dieses Buches gegeben werden.

In Kapitel 1 werden die Grundzüge der Logikentwicklung sowohl theoretisch wie auch praktisch dargestellt. Hierfür wird eine kurze Einführung in das Dualsystem, in die Logik und deren Gesetze, in die Schaltlogik und die Realisierung solcher Schaltungen mit ICs gegeben.

Kapitel 2 versucht möglichst knapp die Struktur und Funktionsweise von Prozessoren und Computern zu erklären (etwas fundierter und ausführlicher als in dieser Einleitung). Auch

die innere Struktur des 68000, der im Atari ST Verwendung findet, wird hier behandelt. Am Anfang dieses Kapitels steht ein kurzer Abriß der geschichtlichen Entwicklung dieses noch jungen Gebiets.

In Kapitel 3 werden die verschiedenen Möglichkeiten des Datentransports sowohl innerhalb wie außerhalb der Computer erläutert. Gerade in den letzten Jahren ist die Vernetzung von kleinen Computern immer mehr in den Vordergrund gerückt. Damit lassen sich Netze aufbauen, die in der Rechenleistung so machen Mittel- oder Großrechner Konkurrenz machen können. Gerade in kleinen und mittleren Betrieben stellen diese Rechnernetze eine preiswerte Alternative zur sogenannten mittleren Datentechnik dar.

In Kapitel 4 wird anhand der im Atari ST und auf der Interfaceplatine integrierten Chips eine Einführung in die Mikroprozessorbausteine gegeben. Dabei werden sowohl der Zentralprozessor MC 68000 wie auch die anderen allgemein käuflichen ICs des Atari ST behandelt, also der WD 1772 (FDC, Floppy Disk Controller, Disketten-Kontroll-Baustein), der MPF 68901 (Multi Function Peripheral, Multifunktions-Peripherie-Baustein), der ACIA 6850 (asynchronous Communications Interface Adapter, serieller I/O-Baustein), der YM 2149 (Sound-Baustein). Daneben werden auch die beiden Bausteine der Interface-Karte beschrieben: der 8255 (paralleler I/O-Baustein) und der 8251 (serieller I/O- Baustein).

Kapitel 5 beschäftigt sich mit der Funktionsweise und dem Aufbau der Hardware des Atari ST. Hier werden auch die Custom-IC des Atari ST behandelt: MMU, DMA, Shifter, Blitter und Glue.

Kapitel 6 gibt eine Einführung in die Verarbeitung analoger Daten. Immer wenn mit einem Computer gemessen oder gesteuert werden soll, wird dieses Thema berührt, da viele Meß- und Steuersignale analog sind, der Computer aber nur digitale Werte verarbeiten kann. Es muß dann also eine Analog-Digital- oder eine Digital-Analog-Wandlung durchgeführt werden. Kapitel 6 stellt verschiedene Prinzipien und Schaltungen für diese Wandelverfahren vor.

Kapitel 7 führt in die Programmierung des Atari ST ein. Bei der Einführung in die Assemblersprache wird noch einmal auf die Struktur des MC 68000 eingegangen, soweit sie für einen Programmierer von Interesse ist.

In Kapitel 8 werden zunächst der Aufbau und die Funktionsweise der beiliegenden Interfacekarte erläutert. Anschließend folgt eine genaue Beschreibung des Aufbaus und der Funktionsweise der Beispielschaltungen für die Interfacekarte.

Ein Anhang mit Anleitungen und Hilfen zur Fehlerbeseitigung, falls die Karte nach dem Bestücken nicht sofort funktioniert, Hinweisen zum Löten für alle, die noch nicht so viel Bastelerfahrung haben, einer Liste aller Befehle des 68000, einer ASCII-Tabelle, den DIN-Symbolen für Flußdiagramme, einer Bezugsquelle für weitere Interfacekarten sowie einem Literaturverzeichnis runden das Buch ab.

Die Autoren wünschen allen Lesern viel Erfolg und Spaß beim Lesen und vor allem gutes Gelingen beim Basteln mit der Interfacekarte.

1 Grundzüge der Logikentwicklung

Die Dinge der Welt müssen so betrachtet werden, als ob sie von einer höchsten Intelligenz ihr Dasein hätten.
Kant

1.1 Logische Verknüpfungen, binäre Zustände

Nahezu alle Rechnersysteme enthalten Komponenten, die lediglich zwei logische Zustände einnehmen können. In der Elektronik kann man diese beiden Zustände als stromführend oder stromlos, als spannungsführend oder spannungsfrei bezeichnen. Zwischen diesen beiden Zuständen existieren alle Übergangspegel nur extrem kurzzeitig, so daß diese elektronischen Funktionen nur durch die zwei digitalen Zustände charakterisiert werden. Ein spannungsführender Pegel wird häufig auch mit dem Symbol »1« oder »H« (von High), ein spannungsfreier Pegel mit »0« oder »L« (von Low) gekennzeichnet.

Für den Aufbau eines Zahlensystems kann man – ohne weitere Kenntnis der elektronischen Hintergründe – die Ziffern 0 und 1 als Basis verwenden. Man verwendet so das Binärsystem (oder Dualsystem), bei dem alle möglichen Zahlen nur aus diesen beiden Ziffern gebildet werden. Die Zahlen im Dualsystem sind im Unterschied zum Zehnersystem sehr lang. Dagegen lassen sich alle Rechenoperationen (wie Addition, Multiplikation usw.) erheblich leichter durchführen. Bild 1.1 zeigt den allgemeinen Aufbau von Zahlensystemen an den beiden Beispielen des dezimalen und binären Systems.

Bei der Umformung einer Dezimalzahl in das binäre Zahlensystem geht man wie folgt vor: Man sucht die höchstmögliche Potenz von 2 auf, die gerade noch in die Dezimalzahl paßt, und subtrahiert diese vom Anfangswert. Sofern diese vorliegende Dezimalzahl > 0 ist, notiert man eine 1. Danach subtrahiert man schrittweise alle folgenden kleineren Zweierpotenzen und notiert jeweils ebenfalls eine 1, wenn die Subtraktion keinen negativen Ausgang erhält. Sofern der Ausgang der Subtraktion negativ wird, notiert man eine 0 und macht die Subtraktion rückgängig. Dieses Verfahren wird mit der Subtraktion von 1 aus dem verbleibenden Rest abgebrochen. Binärzahlen verwandelt man in Dezimalzahlen, indem jeder 1 diejenige Dezimalzahl zugeordnet wird, die ihrem Stellenwert entspricht, und dann alle Dezimalzahlen addiert werden.

Dezimalsystem				Dualsystem				
10^3	10^2	10^1	10^0	2^4	2^3	2^2	2^1	2^0
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	1
0	0	0	2	0	0	0	1	0
.	.	.	.	.	.	.	.	.
0	0	0	9	0	1	0	0	1
0	0	1	0	0	1	0	1	0
.	.	.	.	.	.	.	.	.
0	0	3	1	1	1	1	1	1

Bild 1.1: Beispiele für Zahlen aus dem Dezimal- und Dualsystem

An zwei Beispielen sei die Umformung vom Binär- ins Dezimalsystem bzw. umgekehrt dargestellt:

```
11110111111 = 1024 + 512 + 256 + 128 + 32 + 16 + 8 + 4 + 2 + 1
            = 1983
```

oder umgekehrt:

```
1200        = 1024 + 128 + 32 + 16
            = 10010110000
```

Die Rechenregeln innerhalb der vier Grundrechenarten sind genau wie im Dezimalsystem anzuwenden. Folgende Beispiele verdeutlichen diese Operationen (links dezimal, rechts dual):

Addition

```
      11          1011
      12          1100
       9          1001
 ________     ________
      32        100000
```

Subtraktion

```
      14          1110
-      7          0111
 ________     ________
       7          0111
```

Multiplikation

```
    5*6          101*110
 ------       ----------
     30           110
                    110
              ----------
              11110
```

Division

```
15:3 = 5      1111:11 = 101
              11
              --
               01
                00
                --
                11
```

Dezimal	Dual	Hexadezimal	Oktal
00	00000	00	00
01	00001	01	01
02	00010	02	02
03	00011	03	03
04	00100	04	04
05	00101	05	05
06	00110	06	06
07	00111	07	07
08	01000	08	10
09	01001	09	11
10	01010	0A	12
11	01011	0B	13
12	01100	0C	14
13	01101	0D	15
14	01110	0E	16
15	01111	0F	17

Bild 1.2: Gebräuchliche Zahlensysteme

Wie bei allen Zahlensystemen, erhält man durch Anhängen einer Null an eine Zahl eine Multiplikation um den Basiswert. Im Dezimalsystem ergibt sich also beim Anhängen einer Null der zehnfache, im Binärsystem der zweifache Wert. Streicht man die letzte Zahl rechts, so ergibt sich im Binärsystem eine Division durch 2. Die Sprechweise für binäre Ausdrücke ist

durch die Angabe von Nullen und Einsen wegen der Länge der Zahlen nicht üblich. Es hat sich eingebürgert, eine bestimme Anzahl von Ziffern in einer Gruppe zusammenzufassen. Bei größeren Rechnern oder Minirechnern werden Dreierbündel, bei Mikrorechner Viererguppen verwendet. Bild 1.2 zeigt einen Vergleich der gebräuchlichen Zahlensysteme der Dezimalzahlen von 0 bis 15.

Damit ergibt sich als Sprechweise für die Binärfolge 11010101:

- im Oktalsystem 325
- und im Sedezimalsystem (hexadezimal) D5

Da bei manchen Zahlen nicht eindeutig ist, zu welchem Zahlensystem sie gehören, wird der Oktalzahl ein O und der Sedezimalzahl ein H angehängt oder ein $ vorangestellt (z.B. 3Ah oder $3A). In der Literatur zum Atari ST wird im allgemeinen die Darstellung mit dem vorangestelltem Dollar $ verwendet, weshalb sie auch in diesem Buch verwendet werden soll. Zur Verdeutlichung erhält die Dezimalzahl oft noch einen Punkt. Sowohl das oktale als auch das sedezimale System haben den Vorteil, daß alle Kodierungen des Binärsystems möglich sind. Sogenannte Pseudotetraden wie im Dezimalsystem treten nicht auf (1001 ergibt im Dezimalsystem 9; 1011 dagegen hat keine Zifferndarstellung, sondern entspricht der Zahl 11).

Bei Rechneranwendungen ist das Bit die kleinste Informationseinheit. Ein Bit kann je nach Zustand der logischen Funktion nur die beiden Werte 0 oder 1 annehmen. Eine Kombination von 8 Bit bezeichnet man als Byte und eine Kombination von 16 Bit machen ein Wort aus. Für die Darstellung aller Möglichkeiten innerhalb einer Anzahl von Bits sind verschiedene Kodierungen gebräuchlich. Am weitesten verbreitet ist wohl der Dual-Code, der neben anderen Kodierungen im Bild 1.3 gezeigt ist.

Dezimal	Dual	Gray-Code	Aiken-Code
0000	0000	0000	0000
0001	0001	0001	0001
0002	0010	0011	0010
0003	0011	0010	0011
0004	0100	0110	0100
0005	0101	0111	1011
0006	0110	0101	1100
0007	0111	0100	1101

Bild 1.3: Binäre Darstellung verschiedener Codesysteme

Neben dem Dual-Code wird häufig der Gray-Code verwendet, bei dem sich nur jeweils ein Bit ändert. Beim Übergang von einer zur nächsten Information kann damit ein fehlerhaftes Bit erkannt werden. Der Aiken-Code ist stellenbewertet und komplimentär. So läßt sich das Neunerkomplement durch Vertauschung von 0 und 1 erreichen.

Die Funktion aller komplexen digitalen Systeme läßt sich nur durch das Zusammenspiel logischer Bauelemente und digitaler Komponenten erklären. Alle logischen Schaltungen bestehen aus logischen Elementen, die in ihrer Kombination erst die richtige Funktion ausmachen. Das gilt sowohl für einfachste Schaltungen wie Alarmanlagen oder Pegelüberwachungen als auch für komplexe Digitalrechner. Sofern der Algorithmus der Funktion (also die richtige logische Bedingung) eines Systems genau definiert ist, braucht man nur noch eine logische Schaltung zu finden, welche die geforderte Bedingung erfüllt.

Zum Entwurf logischer Schaltungen bedient man sich logischer Grundfunktionen, um die richtige Gesamtfunktion zu bestimmen. Die grundsätzliche Problemstellung für einen Ausgang läßt sich allgemein durch die logische Verknüpfung darstellen. In der Regel hat man dabei eine Anzahl von Eingangsgrößen und eine Ausgangsgröße. Alle Eingangsgrößen können nur die Werte 0 oder 1 annehmen und ergeben in geeigneten Kombinationen die Ausgangsgröße (Bild 1.4).

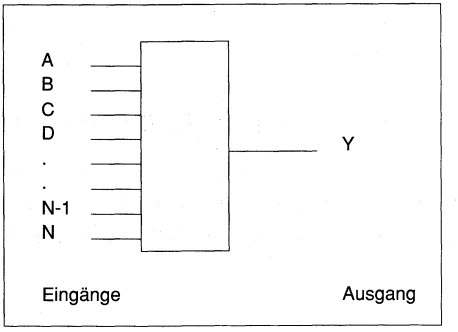


Bild 1.4: Logische Funktionen aus N Eingangs- und einer Ausgangsgröße

Die wichtigsten logischen Funktionen zeigt Bild 1.5 Aus der Negation, der logischen UND- oder der ODER-Funktion lassen sich alle möglichen logischen Schaltungen aufbauen.

logisches UND	$Y = A \wedge B$
logisches ODER	$Y = A \vee B$
Negation	$Y = \overline{A}$

Bild 1.5: Drei der wichtigsten logischen Funktionen: UND, ODER, Negation

1.2 Wahrheitstafeln, Symbole

In vielen Fällen wird für logische Verknüpfungen eine Wahrheitstafel verwendet, die einer Anzahl von Eingängen den jeweils zugehörigen logischen Ausgang zuordnet. Hierbei ist es unbedingt notwendig, alle Kombinationen der Eingänge einzutragen. So gibt es bei zwei Eingängen vier, bei drei Eingängen acht Kombinationsmöglichkeiten (2^n Kombinationen). Für zwei Eingänge sind in Bild 1.6 für das logische UND und das logische ODER die Wahrheitstafeln angegeben.

Eingänge	UND	ODER
A B	Y	Y
0 0	0	0
0 1	0	1
1 0	0	1
1 1	1	1

Bild 1.6: Wahrheitstafel für das logische UND und ODER für zwei Eingänge

Bild 1.7 zeigt die Zuordnung von Eingangs- und Ausgangssignal bei einer Negation. Die logischen Schaltungen werden nach DIN 40700 durch die Schaltzeichen aus Bild 1.8 dargestellt.

Eingang	Ausgang
A	Y
0	1
1	0

Bild 1.7: Die Wahrheitstafel für die Negation

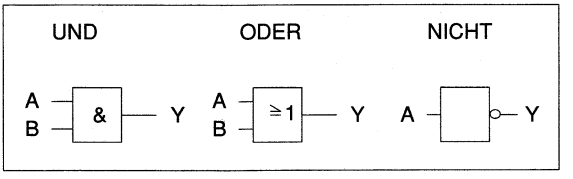
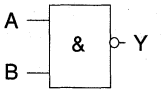


Bild 1.8: Schaltzeichen der logischen Grundfunktionen

Praktisch werden auch direkte UND/ODER-Verknüpfungen mit direkter Negation verwendet. Diese UNDNICHT- oder ODERNICHT-Funktionen sind in viele Logikfamilien integriert und reichen zum Aufbau einer Schaltung aus. Funktion und Wahrheitstafel für das UNDNICHT (engl. NAND) zeigt Bild 1.9.



UNDNICHT (NAND)

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

Bild 1.9: Wahrheitstafel für die UNDNICHT (NAND)-Funktion

1.3 Vereinfachungs- und Umformungsmethoden

Wegen der Ähnlichkeit der Rechenregeln für die Addition und Multiplikation wird in vielen Fällen statt dem logischen UND (^) auch ein * und statt dem logischen ODER (v) ein + geschrieben. Es gelten die folgenden Gesetze zur Vereinfachung der logischen Gleichungen:

1. Grundfunktionen

$A + \underline{A} = A$ $\qquad$ $A * \underline{A} = A$

$A + \overline{A} = 1$ $\qquad$ $A * \overline{A} = 0$

$A + 0 = A$ $\qquad$ $A * 0 = 0$

$A + 1 = 1$ $\qquad$ $A * 1 = A$

2. Kommutativgesetz

$A + B = B + A$ $\qquad$ $A * B = B * A$

3. Assoziativgesetz

$A + B + C = A + (B + C) = (A + B) + C$

$A * B * C = A * (B * C) = (A * B) * C$

4. Distributivgesetz (keine Analogie in der Algebra)

$A * B + A * C = A * (B + C)$

$(A + B) * (A + C) = A + B * C$

Wie schon vorher erwähnt, ist die Benutzung von logischen UND- oder logischen ODER-Elementen zusammen mit der Negation zum Logikentwurf allein ausreichend. Entsprechend

lassen sich in allen Schaltungen ODER-Elemente durch UND-Elemente ersetzen und umgekehrt. Diese Tatsache kann man als de Morgansche Regel formulieren:

$$\overline{A + B} = \overline{A} * \overline{B} \qquad \overline{A * B} = \overline{A} + \overline{B}$$

Aus einer angegebenen Wahrheitstafel schreibt man die gesamte logische Gleichung auf, indem man gleichartige Ausgangszustände jeweils mit ODER spaltenweise und mit UND zeilenweise verknüpft. Es ist dabei gleichgültig, ob als Ausgangsgröße logische Nullen oder Einsen benutzt werden. Wichtig ist nur, daß alle möglichen Kombinationen dargestellt sind (Bild 1.10).

Eingänge			Ausgang
A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Bild 1.10: Beispiel einer logischen Funktion für drei Eingänge und einen Ausgang

$$Y1 = \overline{ABC}, \quad Y2 = \overline{ABC}, \quad Y3 = \overline{A}BC, \quad Y4 = A\overline{BC}$$

insgesamt ergibt sich:

$$Y = \overline{A}*\overline{B}*\overline{C} + \overline{A}*\overline{B}*C + \overline{A}*B*C + A*\overline{B}*\overline{C} \ .$$

Durch Vereinfachung erhält man:

$$\begin{aligned} Y &= \overline{B}*\overline{C}\ (\overline{A}+A) + \overline{A}*C\ (\overline{B}+B) \\ &= \overline{B}*\overline{C}*1 + \overline{A}*C*1 \\ &= \overline{B}*\overline{C} + \overline{A}*C \end{aligned}$$

Sofern bei einer Wahrheitstafel mehr logische Einsen als logische Nullen im Ausgang erscheinen, wird die Darstellung der Gesamtgleichung durch Angabe derjenigen Gleichungen, die eine Null enthalten, vereinfacht. Es ist dann jedoch notwendig, die Funktion zu negieren.

Manchmal sind Ausgangs- oder Eingangsgrößen in einzelnen Funktionen beliebig. In der Wahrheitstafel kann eine solche Größe durch die Variable x dargestellt werden. An derartigen Stellen vereinfacht sich die logische Gesamtgleichung, da hier sowohl eine 0 oder eine 1 einsetzt werden kann.

1.4 Die Methode von Karnaugh und Veitch

Für komplizierte Schaltungen ist die Angabe von Wahrheitstafeln nur für einzelne Teilfunktionen zu durchschauen. Komplexe Zusammenhänge bei logischen Zuständen oder Abläufen können durch solche Tafeln zwar stets dargestellt werden, die Angabe der Logik ist jedoch oft schwierig und selten auf die einfachste Form zu bringen. Es ist deshalb oftmals leichter, die logischen Zustände in einem Diagramm darzustellen und damit zusammenhängende logische Bezirke zu erkennen. Bei einer solchen Karnaugh-Veitch-Tafel werden alle möglichen gewünschten Ausgangsgrößen in Felder eingetragen. Als Spalten- bzw. Zeilenangaben stehen dabei die Eingangsgrößen. Ein KV-Diagramm für zwei Eingänge sieht damit so aus, wie in Bild 1.11 gezeigt.

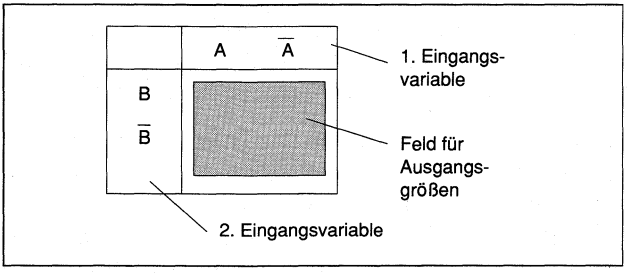


Bild 1.11: Eine Karnaugh-Veitch-Tafel für zwei Eingänge

Bild 1.12 veranschaulicht das KV-Diagramm für ein logisches UND- beziehungsweise ODER-Gatter. Drei Eingangsgrößen lassen sich in einem KV-Diagramm ebenfalls darstellen (Bild 1.13).

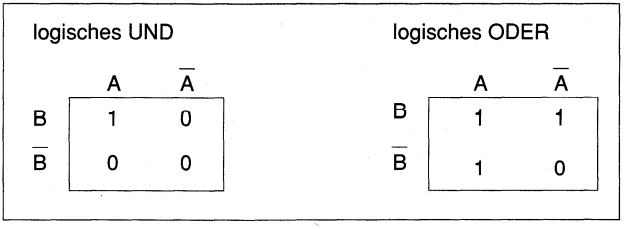


Bild 1.12: Logische UND und ODER in der Darstellung einer Karnaugh-Veitch-Tafel

Auf eine größere Zahl von Eingängen kann man das KV-Diagramm beliebig erweitern. Es ist lediglich notwendig, daß benachbarte Funktionen sich nur in einer Eingangsgröße unterscheiden.

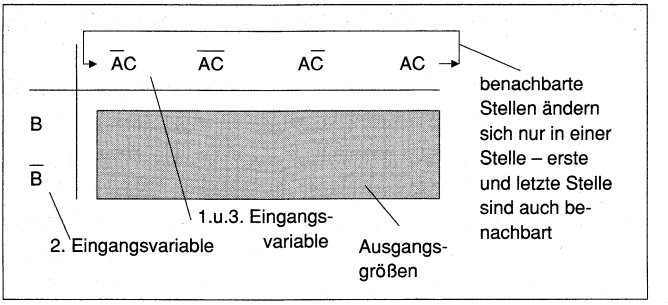


Bild 1.13: Die Karnaugh-Veitch-Tafel für drei Eingänge

Bei der grafischen Anordnung der Ausgangsgrößen in einem Diagramm können überflüssige (sog. redundante) Informationen sofort erkannt werden. Im allgemeinen ist dieses Erkennen von Redundanzen in Gleichungen aus Wahrheitstafeln erst nach längerer Rechnung möglich.

Zur Angabe der logischen Funktion werden im KV-Diagramm zusammenhängende Bezirke aufgesucht und deren Wahrheitsgehalt bezogen auf die Eingangsgrößen angegeben. Bezirke werden auch dann noch als zusammenhängend angesehen, wenn sie den rechten Rand erreichen und sich links fortsetzen (diese Eingangsgrößen unterscheiden sich ja auch nur um einen Wert). Gleiches gilt auch für die Begrenzungen nach oben oder unten. Besonders einfach wird die Ausdehnung eines Bezirks, wenn beliebige Möglichkeiten (also x statt 0 oder 1) vorliegen. Derartige Zonen können nämlich beliebigen Bezirken zugeschlagen werden. Das folgende Beispiel gibt das Erkennen einer Redundanz mit der richtigen logischen Darstellung an. Die Wahrheitstafel dazu zeigt Bild 1.14.

Eingänge			Ausgang
C	B	A	Y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Bild 1.14: Beispiel einer Wahrheitstafel für die Berechnung nach der Karnaugh-Veitch-Methode

Aus dieser Tafel folgt:

$$
\begin{aligned}
Y &= \bar{A}*\bar{B}*\bar{C} + \bar{A}*B*\bar{C} \\
&= \bar{A}*\bar{C}\ (\bar{B}+B) \\
&= \bar{A}*\bar{C}
\end{aligned}
$$

Bei dieser Logik war also die Eingangsvariable B redundant. In einem KV-Diagramm kann man diesen Sachverhalt sofort ablesen. Für dieses Beispiel ergibt sich nämlich das Diagramm wie Bild 1.15 gezeigt.

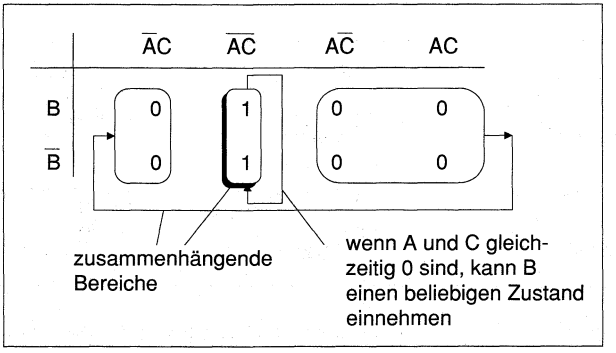


Bild 1.15: Überflüssige Bereiche in einer Wahrheitstafel nach Karnaugh-Veitch

Für mehr als vier Eingangsvariablen (z.B. sechs) läßt sich das KV-Diagramm auch räumlich darstellen. Dabei ordnet man jeweils vier Flächen mit 16 Elementen (entsprechend vier Eingängen) in vier Ebenen übereinander an.

1.5 Logische Schaltzeichen, Logikfamilien

Die Schaltzeichen für UND-, ODER- und NAND-Elemente wurden bereits vorgestellt. Nach DIN haben dabei alle logischen Elemente rechteckige Formen und tragen das Logiksymbol im Inneren (Bild 1.16).

Sofern sich am Ausgang ein Kreis befindet, hat der Ausgang eine negierte Funktion. Die Darstellung nach DIN ist für nichtgrafikfähige EDV-Systeme wegen der leichteren Darstellung von Vorteil. Da das Schaltbild bei dieser Norm sehr unübersichtlich wird, benutzt man beim Schaltungsentwurf oft noch die früher in Europa verwendeten Schaltzeichen oder die nach der amerikanischen Norm, bei der Gatter sofort aufgrund ihrer Form von Flipflops zu unterscheiden sind (Bild 1.17).

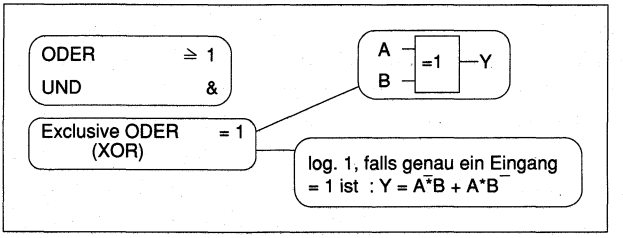


Bild 1.16: Logiksymbole nach DIN

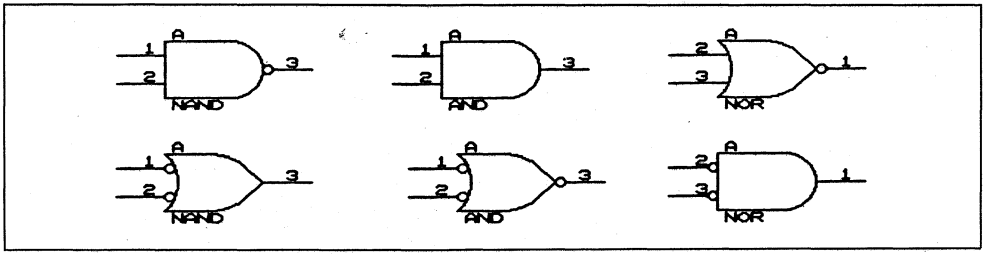


Bild 1.17: Darstellung NAND-, AND- und NOR-Gatter in der amerikanischen Norm

Zum Aufbau von Schaltungen stehen dem Benutzer zahlreiche Logikfamilien zur Verfügung, mit deren Hilfe beliebige Schaltungen realisiert werden können. Diese Logikfamilien unterscheiden sich in der Herstellungstechnologie. Die elektrischen Eigenschaften sind deshalb je nach Familie recht unterschiedlich. Im Mikrocomputerbereich werden vorwiegend Bausteine aus drei Logikfamilien verwendet. Mikrobausteine und Peripheriebausteine sind fast ausschließlich in NMOS-Technik ausgelegt. Diese Technologie wurde von der Firma Intel entwickelt und bildet die Grundlage nahezu aller Mikroelemente. NMOS-Bausteine kommen in der Regel mit einer Betriebsspannung von fünf Volt aus und brauchen Ströme bis zu 100 mA pro Baustein (typischer Wert). In NMOS sind die Durchlaufzeiten der Gatter hinreichend kurz, was eine hohe Datenrate bewirkt. Da NMOS eine hohe Integrationsdichte zuläßt, können hiermit hochintegrierte Bauteile hergestellt werden.

Auf der Basis der CMOS-Technologie gibt es zahlreiche Bausteine, die sowohl logische Elemente als auch Mikroelemente integriert haben. CMOS ist erheblich flexibler in der Versorgungsspannung und benötigt nahezu nur Strom während interner Schaltvorgänge. In den letzten Jahren hat die CMOS-Entwicklung entscheidende Fortschritte gebracht, so daß heute auch kurze Schaltzeiten möglich sind. Die Störempfindlichkeit von CMOS ist extrem niedrig und Speicherelemente lassen sich wegen der geringen Stromaufnahme mit einer Batterie über Jahre hinaus puffern.

Wegen der Vielfältigkeit der Elemente, der kurzen Durchlaufzeit und den ausgezeichneten Eigenschaften kann man heute jedoch kaum auf die TTL-Logikfamilie verzichten. Diese Familie wird mit fünf Volt betrieben und nimmt in der LS-Version ca. 2 mA pro Gatter auf. Die typische Durchlaufzeit liegt bei ca. 10 ns und erlaubt damit eine Verarbeitungsfrequenz bis 40 MHz. Die Elemente dieser Logikfamilie sind in Keramik- oder Plastikgehäusen untergebracht und es gibt für alle Standardanwendungen wie UND, ODER, XOR, Inverter, Flipflops usw. spezielle Bausteine. Jeder, der elektronische Schaltungen entwickelt, sollte deshalb über den Aufbau, die Vielfalt und die Eigenschaften dieser Elemente informiert sein. Der Eingang eines Gatters in LSTTL-Technik braucht ca. 200 µA zum Schalten und wird der Einfachheit halber mit FAN-IN = 1 bezeichnet. Die Zahl des FAN-OUTs gibt dann an, wie viele Eingänge (vom FAN-IN=1) betrieben werden können. Offene Eingänge bei TTL sind logisch 1. Bild 1.18 gibt eine Übersicht über die Eigenschaften der wichtigsten Logikfamilien.

Logikfamilie	Durchlaufzeit	Leistungsaufnahme	
	in ns/Gatter	in mW/Gatter	typische Versorgung
TTL	10	10	5-Volt-Versorgung
TTL-LS	10	2	
TTL-ALS	6	1	
TTL-S	3	20	
ECL	2	25	+2, –3-Volt-Versorg.
NMOS	10-50	0,1	5-Volt-Versorg.
HCMOS	20-50	0,1-0,01	
CMOS	50-350	0,01	3- bis 18-Volt-Versorg.

Bild 1.18: Die wichtigsten Eigenschaften der TTL-Familie

Neben der LSTTL-Serie gibt es auch noch Standard-TTL-, Schottky-TTL- und ALS-TTL-Elemente. Schottky-TTL ist erheblich schneller als TTL (ca. 2 bis 5 ns Durchlaufzeit), benötigt dafür aber mehr Strom. Low-Power-Schottky (LSTTL) ist genau so schnell wie TTL, benötigt jedoch um den Faktor 2 bis 10 weniger Betriebsstrom. Advanced Low-Power-Schottky (ALS) kommt mit noch weniger Strom aus als LSTTL und ist nahezu doppelt so schnell.

TTL ist eine Abkürzung für Transistor-Transistor-Logik, MOS ist eine Abkürzung für Metal-Oxide-Silicium und ECL steht für Emitter-Coupled-Logik. HMOS ist eine MOS-Familie für hohe Geschwindigkeit. CMOS ist eine Abkürzung für Complementary-MOS. Weitere technische Eigenschaften werden bei den folgenden Anwendungen vorgestellt.

1.6 Sequentielle logische Schaltungen

Bei vielen Schaltungen benötigt man neben den statischen Elementen auch solche, deren Zustände von der Vorgeschichte abhängen. Mit derartigen Schaltungen gelingt es, kurzzeitige Impulse festzuhalten, Frequenzen abzuzählen oder Zustände beliebig lange zu speichern.

Die Schaltung in Bild 1.19 besteht aus zwei NAND-Gattern und hat die Eigenschaft, einen logischen 0-Impuls abzuspeichern. Bei derartigen sequentiellen Schaltungen werden die Ausgänge (zur Unterscheidung statischer Schaltungen) mit dem Buchstaben Q bezeichnet.

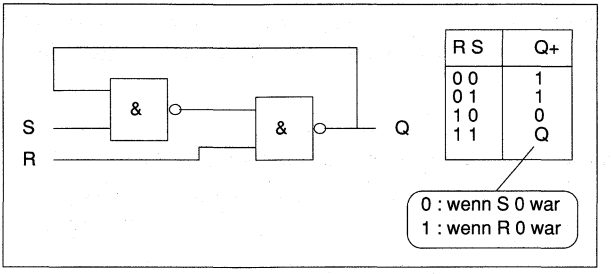


Bild 1.19: Schaltung von zwei NAND-Gattern, wobei der Ausgangszustand von der Vorgeschichte abhängt

Mit Q+ wird der Zustand des Ausgangs bezeichnet, nachdem der logische Übergang stattgefunden hat. Die obige Schaltung ist damit in der Lage, denjenigen Eingang (S oder R) am Ausgang (Q) festzuhalten, der zuletzt mit einer log. 0 angelegen hat. Diese Schaltung kann die Grundlage eines Speicherelements für ein Bit bilden. Man bezeichnet ein solches Element auch als RS-Flipflop (Reset-Set-Flipflop – Rücksetz-Setz-Flipflop). Der Einfachheit halber wird für diese Schaltung auch oft nur das Symbol angegeben (Bild 1.20).

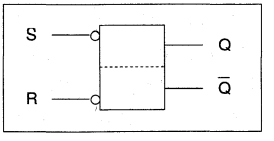


Bild 1.20: Schaltsymbol des RS-Flipflops

Q und Q+ sind zueinander komplementär, wobei niemals gleichzeitig S und R anliegen darf. Die logische Gleichung des RS-Flipflops ist damit:

$$Q+ = \overline{R} + S * Q \qquad (\text{ mit } S + R = 1 \text{ })$$

Das Inversionszeichen am Eingang des Flipflops zeigt, daß nur eine logische 0 als Eingangssteuergröße zugelassen ist.

Ein weiteres wichtiges Element ist das getaktete JK-Flipflop. Bei diesem Element wird durch jeden Taktimpuls an einem Takteingang (Clock) der Ausgang je nach dem Zustand der Eingänge J und K bestimmt. Sofern J und K gleichzeitig logisch 1 sind, ändert der Ausgang mit jeder Taktflanke seinen Zustand. Mit einem solchen Flipflop ist es somit möglich, eine Eingangsfrequenz durch zwei zu teilen. Sind beide Eingänge 0, so bleibt der Ausgang unverändert; ist J=1 und K=0, wird Q=1 und ist K=1 und J=0, so wird Q=0. Diese Aussagen lassen sich in der Gleichung

$$Q+ = J * \overline{Q} + \overline{K} * Q$$

zusammenfassen.

Das Schaltsymbol für ein JK-Flipflop zeigt Bild 1.21.

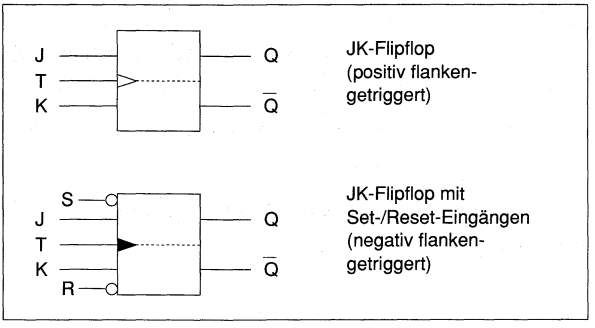


Bild 1.21: Schaltsymbol des JK-Flipflop

Es gibt in jeder Familie Takt-Flipflops, die auf den Übergang von 0 auf 1 oder von 1 auf 0 triggern. Erstere haben im Schaltbild ein weißes, letztere ein schwarzes Dreieck als Takteingang. Ein spezielles JK-Flipflop ist das D-Flipflop, bei dem durch jeden Taktimpuls am Ausgang (Q) der logischen Zustand des Eingangs (D) übernommen wird (Bild 1.22).

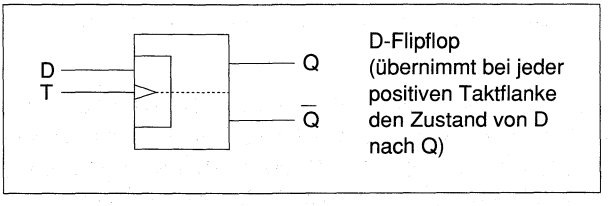


Bild 1.22: Schaltsymbol des D-Flipflop

Das Verhalten eines D-Flipflops läßt sich am einfachsten an einem Timing-Diagramm (Zeitdiagramm) darstellen. Bei einem Timing-Diagramm werden die Ein- und Ausgänge einer Schaltung in eine zeitliche Abfolge gebracht und grafisch dargestellt. In dem Timing-Diagramm in Bild 1.22 befindet sich die Bezeichnung des Ein- oder Ausgangs auf der linken Seite. Rechts daneben folgt die Darstellung des zeitlichen Verlaufs. Die Zeit ist dabei von links nach rechts in positiver Richtung aufgetragen. Für die Schaltung interessante Übergänge oder logische Funktionen sind durch Pfeile (Bild 1.23) gekennzeichnet.

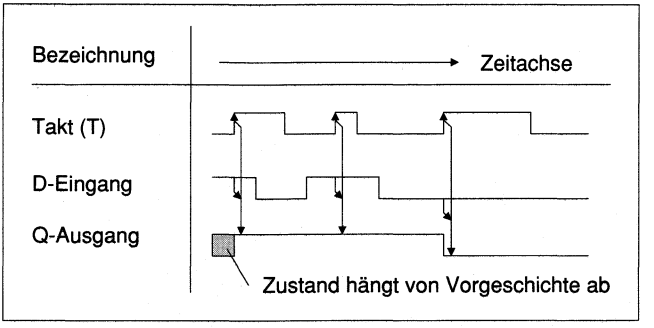


Bild 1.23: Timing-Diagramm für eine Signalfolge an einem D-Flipflop

An zwei Beispielen seien typische Anwendungen von Flipflops vorgestellt. Einen Frequenzteiler durch den Faktor 3 zeigt Bild 1.24.

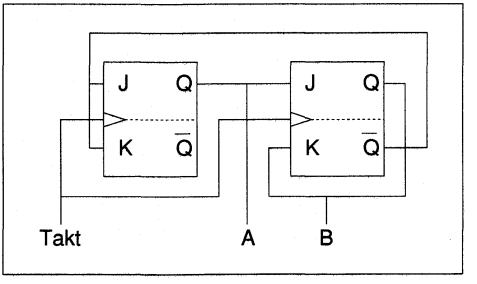


Bild 1.24: Schaltung für einen Frequenzteiler

Um die Schaltung zu verstehen, nehme man an, beide Q-Ausgänge (hier mit A und B bezeichnet) liegen am Anfang auf logisch 0. Am ersten Flipflop befinden sich damit die beiden J- und K-Eingänge zusammen auf logisch 1. Beim ersten Taktimpuls erfolgt damit an diesem Flipflop ein Übergang auf logisch 1 am Ausgang. Das zweite Flipflop ändert seinen Zustand nicht, da vor dem Taktimpuls J und K gleichsam logisch 0 sind. Der nächste (also zweite)

Taktimpuls läßt das erste Flipflop wieder zurückschalten. Da aber nun am zweiten Flipflop J=1 und K=0 vorliegt, ändert sich der Ausgang B ebenfalls. Am ersten Flipflop liegt damit eine logische 0 am J-Eingang und beim folgenden (dritten) Taktimpuls wird wieder die Grundstellung an beiden Flipflops erreicht. Die Zählfolge läßt sich damit so darstellen, wie in Bild 1.25 gezeigt.

B A	
00	K1=1,J1=1;K2=0,J2=0
01	K1=1,J1=1,K2=0,J2=1
10	K1=0,J1=0,K2=1,J2=0
00	

Bild 1.25: Zählfolge der im Bild 1.24 dargestellten Schaltung

Bild 1.26 zeigt ein Schieberegister, bei dem eine 4-Bit-Information mit jedem Takt um eine Stelle nach rechts verschoben wird. Mit einer derartigen Logik läßt sich mit jedem Taktimpuls eine vorgegebene Binärzahl durch den Wert 2 dividieren. Die Schaltung ist aus vier D-Flipflops aufgebaut. Die Funktion kann für ein spezielles Beispiel an der Wahrheitstafel abgelesen werden.

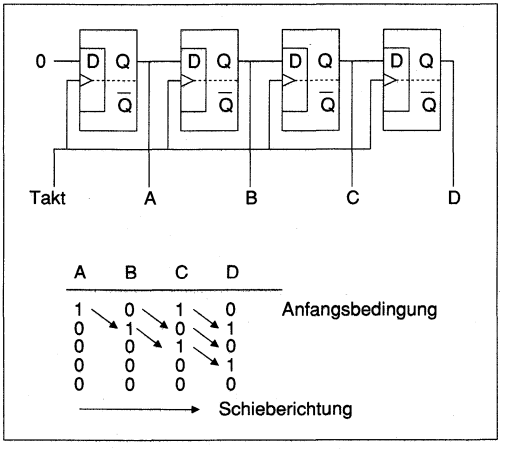


Bild 1.26: Schaltung und Zählfolge für ein Schieberegister. Jede Verschiebung entspricht einer Division durch den Wert 2.

Weitere Schaltungen statischer und sequentieller Logik folgen später bei konkreten Aufgaben aus dem Bereich der Signalverarbeitung innerhalb von Mikrocomputerschaltungen.

2 Aufbau und Funktionsweise von Prozessoren und Systemen

Sag mir exakt, worin deiner Meinung nach der Mensch einer Maschine überlegen sei, und ich werde einen Computer bauen, der deine Meinung widerlegt.
Alan M. Turing

2.1 Geschichtlicher Überblick

Mikroprozessoren und Mikroprozessorsysteme sind eine Konsequenz aus der Rechnerentwicklung und dem Fortschritt der Halbleitertechnologie. Dabei wurde das Grundwissen aus dem Computerbau und der Softwaretechnik der letzten 30 Jahre mit der modernen Integrationstechnik der sechziger und siebziger Jahre verbunden. Der Mikroprozessor bietet auf kleinstem Raum nahezu alle Eigenschaften von Großrechnern der letzten Jahrzehnte, hat jedoch darüber hinaus ganz entscheidende Vorteile. Neben extrem niedrigen Kosten, geringerem Stromverbrauch und vielfältigerer Einsatzweise zeichnet sich der Mikroprozessor insbesondere durch eine hohe Arbeitssicherheit aus. Durch die Integration auf einem Chip fallen zahlreiche Fehlerquellen wie Lötverbindungen, Kontakte usw. weg. Von einem Mikroprozessor werden im Mittel mehr als 100 Milliarden Operationen ohne Fehler durchgeführt. Die Arbeitssicherheit übersteigt damit diejenige von sonstigen Rechnern um das hundert- bis tausendfache.

Rechenmaschinen sind keineswegs eine Erfindung unseres Jahrhunderts. Bereits 3000 v. Chr. benutzte der Mensch den Abakus als einfachste Rechenhilfe. 1642 wurden von Pascal und 1672 von Leibniz Rechenmaschinen gebaut, welche die vier Grundrechenarten beherrschten. 1835 gelang es Babbage einen Rechner zu bauen, der bereits Peripherie steuern konnte. Babbage verstand es außerdem, auch Programmabläufe flexibel zu halten (er kann damit als Vater des modernen Computers angesehen werden). 1915 entwarf Torres einen Rechner, der Entscheidungen fällen konnte. Nach einigen Verbesserungen dieser Systeme und den ersten Anwendungen von Analogrechnern entwickelte Zuse 1936 erstmalig genaue Vorstellungen von automatischen Rechnern. Er benutzte Binärsysteme und Gleitkomma-Arithmetik. Bereits 1941 stellte Zuse Programmiersprachen vor, die dem heutigen PL/1 und ALGOL als Vorläufer dienten.

Von 1937 bis 1950 wurden dann entscheidende Fortschritte im Aufbau von logischen Schaltkreisen gemacht. 1945 entwickelte von Neumann erstmals das Konzept des flexiblen

Programmspeichers. Der erste Computer kostete ca. 10 Millionen Dollar, er enthielt nahezu 20000 Röhren und war 30 Tonnen schwer.

Technologisch gesehen folgte der erste Schritt zum heutigen Rechner 1948 bis 1951 mit der Erfindung des Transistors (Bardeen, Brattain und Schockely). Der Weg zur Integration und damit zum Mikroprozessor wurde jedoch erst 1958 durch die Erfindung des Planartransistors ermöglicht. Seit dieser Zeit konnten immer mehr Bauteile in unterschiedlichen Technologien auf einem einzigen Chip untergebracht werden. Die Geschichte der Mikroprozessorentwicklung ist eng mit dem Namen der Firma Intel verbunden, die im Jahre 1971 die erste vollintegrierte Zentraleinheit 4004 auf den Markt gebracht hat. Der Nachfolger dieses ersten 4-Bit-Prozessors wurde der 8-Bit-Prozessor 8008, dem im Jahre 1974 der Mikroprozessor 8080 folgte. Der 8080 wurde von vielen Firmen kopiert, in Lizenz nachgebaut oder weiter entwickelt. Einer der Weiterentwicklungen ist der Prozessor Z80 von ZILOG. Sehr viele Mikrocomputer der 8-Bit-Generation waren mit einem Prozessor aus dieser Linie ausgerüstet, nicht zuletzt, weil hierfür erstmals ein komfortables Betriebssystem unter dem Namen CP/M (Control Programm for Microcomputers) von Digital Research angeboten wurde.

Im Jahr 1978 erschienen die ersten 16-Bit-Prozessoren. Während die Firma Intel die Prozessoren der Typen 8086 (8088), 80186, 80286 und 80386 als Basis der IBM-Rechner herausbrachte, entwickelte Motorola vollkommen neue Zentraleinheiten mit der Bezeichnung 68000, 68010, 68020 und 68030. Im Gegensatz zur Intel-Strategie waren die Prozessoren deutlich leistungsfähiger als die entsprechenden 8-Bit-Zentraleinheiten. So gelang es zwar mit diesen Prozessoren der 68xxx-Serie ein völlig neues Befehlskonzept zu schaffen, dafür mußten aber alle Programme neu entwickelt werden, da die Kompatiblität zu den alten 8-Bit-Einheiten fehlte.

2.2 Struktur von Zentralprozessoren

Die Struktur der heutigen Rechner geht auf die theoretischen Arbeiten von Burks, Goldstine und von Neumann zurück. Das Ergebnis dieser Studien war ein universelles Konzept eines Rechnerautomaten, der bei stets gleichbleibendem Aufbau unterschiedliche Probleme lösen konnte. Logische Abläufe sollten nicht mehr durch die Vorgabe der logischen Funktion auf das spezielle Problem angepaßt werden, sondern immer gleich bleiben, während die spezielle Funktion durch eine Ablaufsteuerung realisiert werden sollte. Der physikalische Aufbau eines solchen Rechners enthält damit alle Komponenten, die zur Lösung der verschiedenen Aufgaben notwendig sind. Die Anpassung auf das spezielle Problem geschieht durch ein zusätzliches Programm, das die Funktionsweise der Hardware in ihrem Ablauf bestimmt. Die Struktur der heutigen Zentraleinheiten hat sich seit der Entwicklung dieses Von-Neumann-Konzepts nur noch geringfügig geändert. Unterschiede existieren lediglich in dem Aufbau der Hardware und in der Art und Weise der internen Verarbeitung. An einem einfachen Beispiel soll dieser fundamentale Schritt näher erläutert werden: Aus der Aussagenlogik ist

bekannt, daß es nur weniger Elemente bedarf, jede beliebige (noch so komplexe) Aufgabe zu lösen. Mit einem logischen UND und der NICHT-Funktion alleine lassen sich – bei geeigneter Kombination – alle nur denkbaren Funktionen aufbauen. Die folgende Anwendung macht deutlich, wie ein Rechenautomat ein logisches UND für zwei Eingänge und einen Ausgang realisiert. Von der Hardware her benötigt der Rechner zur Lösung des Problems zwei Eingänge und einen Ausgang. Die logische UND-Funktion wird nicht fest verdrahtet – also durch die Hardware festgelegt –, sondern durch die Software beschrieben. So ist es Aufgabe des Programmierers, einen Algorithmus zu finden, der das Problem richtig beschreibt. Aus dem festgelegten Algorithmus gewinnt man eine schrittweise Abfolge zum Abarbeiten innerhalb der Hardware. In dem vorgestellten Beispiel besteht der Algorithmus darin, die beiden Eingänge abzufragen und den logischen Ausgang nur dann auf logisch 1 zu setzen, wenn beide Eingänge eine logische 1 anzeigen. Das daraus resultierende Programm könnte also im ersten Schritt den Eingang A abfragen. Sollte dieser Eingang bereits eine logische 0 enthalten, so kann der Ausgang – ohne weitere Abfrage des anderen Eingangs – auf logisch 0 gesetzt werden. Sofern A eine logische 1 ethält, muß noch der Eingang B abgefragt werden. Vom Zustand dieses Eingangs hängt dann der Ausgangszustand »Y« ab. Ein Programm für die logische UND-Funktion hat also folgendes Aussehen:

Als Anfangsbedingung sei der Ausgang Y = 0.

1. Schritt: Lese den Wert des Eingangs A
2. Schritt: Falls A = 0 ist, gehe zum 5. Schritt
3. Schritt: Lese den Wert des Eingangs B
4. Schritt: Falls B = 1 ist, setze den Ausgang Y = 1
5. Schritt: Ende

Diese mehr verbale Formulierung der Aufgabe ist in Bild 2.1 rechts als Flußdiagramm dargestellt. Flußdiagramme sind wichtige Hilfsmittel, um ein Problem richtig zu beschreiben. Die häufigsten Symbole für Flußdiagramme sind im Anhang (Abschnitt 9.6) dargestellt. Das Flußdiagramm beginnt an der mit »Anfang« gekennzeichneten Stelle. Als erste Funktion wird der Eingang A gelesen und sein Wert bestimmt. Je nach dem ermittelten Wert von A verzweigt sich das Programm (2. Schritt). Der Wert von B wird nur dann geholt, wenn A logisch 1 war. Sofern nach dem Einlesen von B auch ein logischer 1-Zustand festgestellt wird, erfolgt eine Änderung des Ausgangs.

So einfach dieses Beispiel auch sein mag, so gibt es doch im Überblick die notwendigen Funktionen für ein Rechnersystem wieder. Man benötigt ein Eingangswerk, von dem der interne Prozeß seine Eingangsdaten erhält (im Beispiel: A und B), ein Ausgangswerk, das die gewünschte Funktion realisiert (im Beispiel: Y), das notwendige Programm und ein Speicherwerk, in dem die ermittelten Ergebnisse abgelegt werden. Das Leitwerk hat die Aufgabe, die schrittweise Abarbeitung des Programms durchzuführen, und das Rechenwerk entscheidet über den Ausgand der logischen Operationen, die das Leitwerk vorschreibt (Bild 2.2). Sowohl das Programm als auch die Zwischenergebnisse (Daten) befinden sich in einem gemeinsamen Speicher, was typisch für die Rechner der Von-Neumann-Struktur ist.

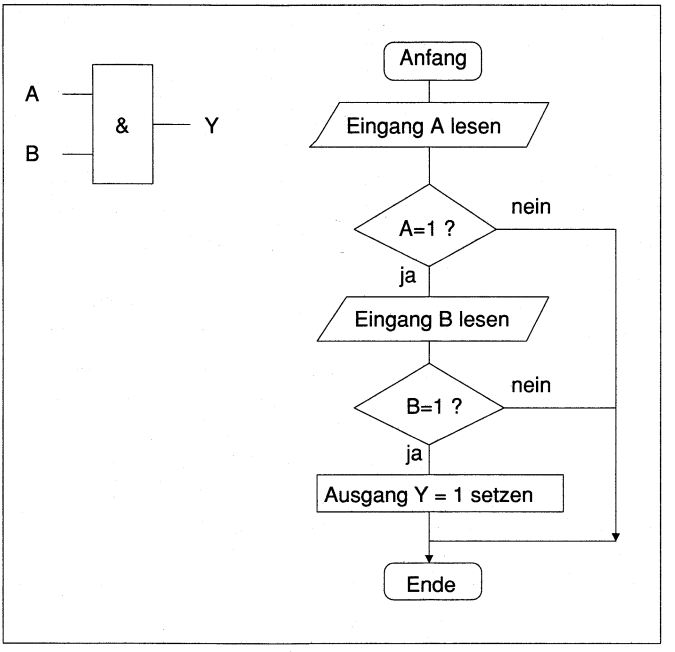


Bild 2.1: Flußdiagramm zum Realisieren einer UND-Funktion

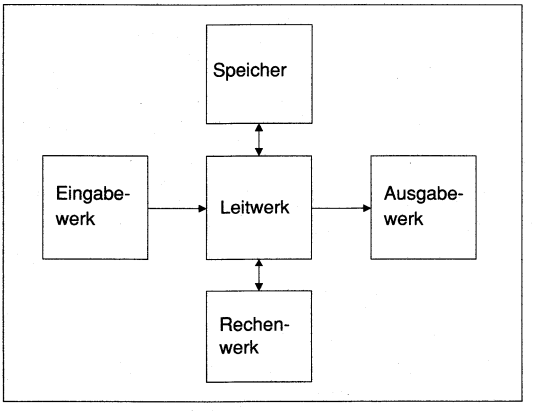


Bild 2.2: Elemente eines Rechners nach der Von-Neumann-Struktur

Für die in dem Programm vorgestellten Schritte hat jeder Rechner genau festgelegte Befehle, die den verbalen Aussagen entsprechen. Die Befehle bestehen dabei aus Bitkombinationen, die entweder byteweise (8 Bits) oder wortweise (16 Bits) zusammengefaßt sind. Je nach der

angegebenen Länge kann damit das Leitwerk 256 (8 Bits) oder 65536 (16 Bits) unterschiedliche Befehle an das Rechenwerk weiterleiten. Der richtige Ablauf bei der Programmverarbeitung läßt sich aus dieser Struktur bereits absehen: Das Leitwerk holt den ersten Befehl (Schritt) aus dem Speicher, interpretiert die gewünschte Funktion und übergibt diese dem Rechenwerk. Das Rechenwerk entscheidet dann über den Ausgang der Operation und legt über das Leitwerk das Ergebnis im Speicher ab. Ist der Befehl vollständig ausgeführt, holt das Leitwerk den nächsten Befehl und führt dessen Operation aus. Diese Vorgänge wiederholen sich, bis das Programmende erreicht wird. Jeder Befehl steht auf einem Speicherplatz mit einer festen Adresse, und ein Programm wird aus hintereinanderliegenden Befehlen in diesem Speicher aufgebaut. Das Leitwerk holt in der Regel die Befehle in aufsteigender Reihenfolge aus dem Speicher und stellt diese dem Rechenwerk zur Verfügung. Es beginnt dabei bei einer festen Adresse und erhöht diese nach jedem abgearbeiteten Befehl um eins. Diese Folge wird lediglich durch Sprungbefehle (wie beim Beispiel im Schritt 2) oder durch das Programmende unterbrochen. Bild 2.3 zeigt die einfachste Verarbeitung bei einer byteweisen Organisation im Speicher.

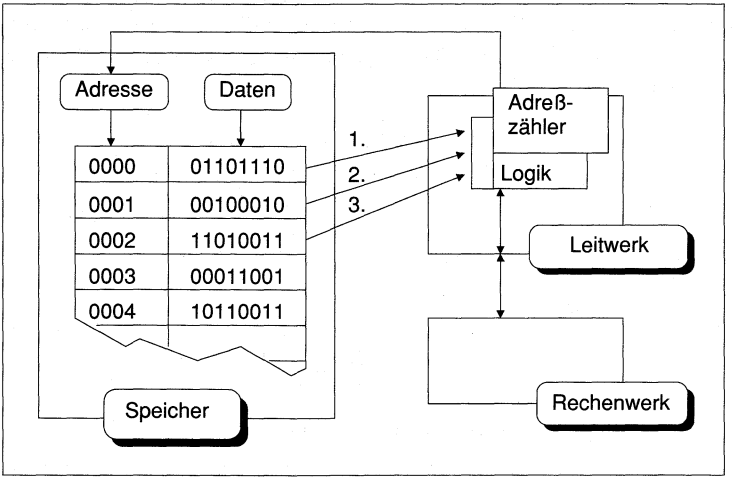


Bild 2.3: Programmablauf in einem Rechner

Leitwerk und Rechenwerk sind bei den heutigen Mikroprozessoren häufig als eine Einheit ausgebildet. Man bezeichnet diesen Komplex auch als Zentraleinheit oder CPU (Central Processing Unit). Vielfach stehen innerhalb dieser Zentraleinheit auch interne Speicherplätze zur Verfügung (Register), die eine Zwischenablage von Werten ohne den externen Speicher ermöglichen. Unabhängig davon, wie die interne Struktur einer Zentraleinheit beschaffen ist, ergibt sich bei der Programmverarbeitung eine fortlaufend abwechselnde Folge von Speicherzugriffen, die Befehle in die CPU holen (OPCODE-FETCH), und Verarbeitungszyklen (EXECUTE) für interne Operationen (Bild 2.4).

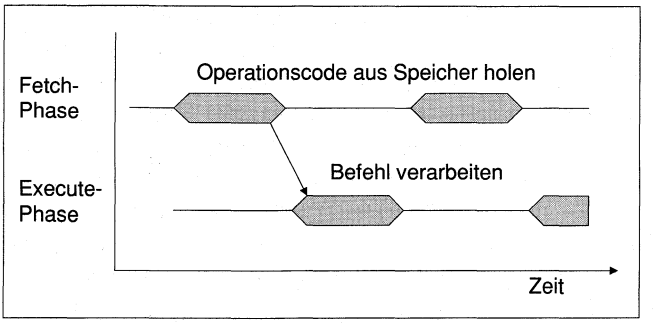


Bild 2.4: Zeitdiagramm bei der Ausführung von Maschinenbefehlen

Das äußere Bild einer Zentraleinheit wird somit durch die Adressiereinheit des externen Speichers (Adreßbus), durch die Befehleinleseeinheit (Datenbus) und durch das Eingabe-/ Ausgabe-Werk bestimmt (Bild 2.5). Im Zusammenhang mit den Verbindungsleitungen einer Zentraleinheit mit ihrer Umwelt wird wegen der speziellen Adreßanwahl und dem Datentransport von einem Bus gesprochen. Die genaue Funktionsweise von Bussystemen ist im Abschnitt 3 beschrieben.

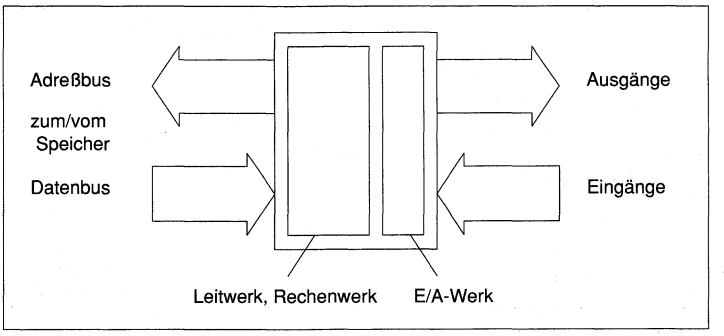


Bild 2.5: Aufbau einer Zentraleinheit in integrierter Form mit den Anschlüssen für die Kommunikation mit der Außenwelt

Alle heutigen Mikroprozessoren entsprechen dieser Struktur. Im Detail haben sich jedoch gewisse Varianten durchgesetzt, die sich sowohl auf die äußere Erscheinung als auch auf die Arbeitsweise auswirken. So dient der Datenbus in der Regel nicht nur zur Entgegennahme der Operations-Codes, sondern vermag auch selbst Veränderungen im Speicherbereich durchzuführen. Er hat damit nicht nur lesende, sondern auch schreibende Funktion. Da ein Großteil der Zentraleinheiten nicht gleichzeitig auf den Speicher zugreifen und das Eingabe-/ Ausgabe-Werk bedienen kann, wurde bei vielen Prozessoren auch die Ein- und Ausgabe

über den Datenbus realisiert. Die CPU verfügt über einen Kontrollbus, der den internen Zustand der CPU nach außen weitergibt. Über ihn wird entschieden, ob der Transfer mit dem Speicher oder einer Ein-/Ausgabe-Einheit stattfindet. Im einfachsten Fall besteht der Kontrollbus aus einer Leseaufforderung (RD: READ), einer Schreibaufforderung (WR:WRITE) und einem Kontrollsignal, das über Speicher- oder Ein-/Ausgabe-Zugriff informiert (Bild 2.6).

$\overline{RD}$	$\overline{WR}$	$\overline{Slct\ 4}$	Funktion
0	0	1	verboten
1	0	1	Speicher schreiben
0	1	1	Speicher lesen
0	1	0	ROM-Port lesen
1	1	1	interne Operation

Bild 2.6: Einfachste Form des Kontrollbusses für ein Rechnersystem

Bild 2.7 zeigt eine Zentraleinheit, wie sie heute in Mikroprozessoren integriert ist. Zur Ansteuerung der Peripherie enthält diese Einheit den Adreßbus, den Datenbus und den Kontrollbus. Neben den vorher genannten Kontrollbusleitungen ist eine weitere Eingangsleitung wichtig, die den Zentralprozessor (z.B. nach dem Einschalten der Betriebsspannung) in einen definierten Grundzustand bringt. Zu diesem Grundzustand gehört es auch, daß der Adreßbus auf die feste Adresse zeigt, von wo der erste Befehl erwartet wird.

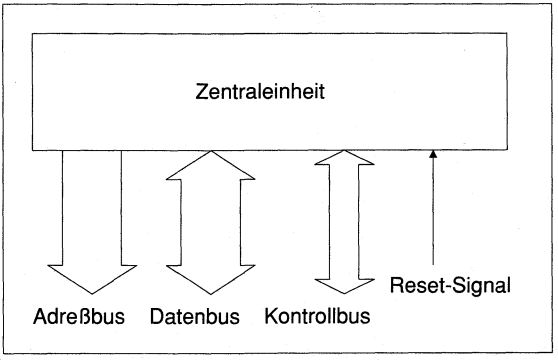


Bild 2.7: Prinzipieller Aufbau des Bussystems eines Computers

Die feste Abfolge jeweils eines FETCH- und eines EXECUTE-Zyklus ist nachteilig, da zahlreiche Zentraleinheiten eine unterschiedliche Anzahl an Bits für unterschiedliche Befehle benötigen. So kommt es vor, daß die CPU in einem byteorientierten Speicher für eine Informa-

tion, die aus 24 Bit besteht, dreimal hintereinander zugreifen muß, ehe eine Verarbeitung erfolgt. Das Rechenwerk innerhalb einer Zentraleinheit führt die logischen und arithmetischen Befehle nach Vorschrift des Leitwerks aus. Bei Mikroprozessoren kann dieses Rechenwerk Operationen nicht nur bitweise, sondern auch byte- oder wortweise durchführen. Neben den logischen Operationen sind auch noch Rechenoperationen wie Schiebebefehle, Komplementbildung, Addition, Subtraktion und teilweise auch Multiplikation und Division implementiert. Das Rechenwerk wird auch mit ALU (Arithmetical and Logical Unit) bezeichnet und spiegelt die Organisationsbreite des gesamten Mikroprozessors wider. So wird der 68000 oft auch als 32-Bit-Prozessor bezeichnet, da er über eine 32-Bit-ALU verfügt. Auch alle anderen Register sind bei ihm 32 Bit breit, nur der aus dem Prozessor herausgeführte Datenbus hat beim 68000 noch ein Breite von 16 Bit. Bild 2.8 zeigt den internen Aufbau einer 16-Bit-ALU. Die logische Funktion wird bei einem derartigen Parallelwerk vom Schaltnetz durchgeführt. Das Schaltnetz erhält den Befehl vom Leitwerk und führt die gewünschte Operation mit den beiden vorliegenden Quellen durch. Das Ergebnis wird in einem Zielregister abgelegt. Eventuelle Besonderheiten bei der Ausführung des Befehls landen in einem Sonderregister (Flag-Register, z.B: Überlauf bei Addition oder ein Ergebnis, das Null ist).

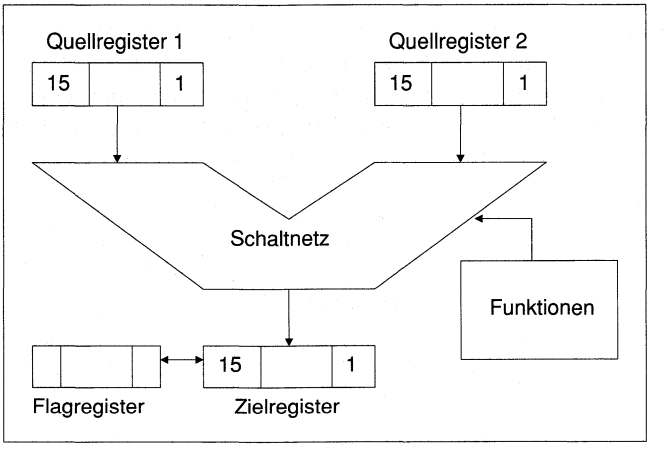


Bild 2.8: Prinzipieller Aufbau der ALU (Arithmetical and Logical Unit) einer CPU

Während die ALU mit Daten aus dem Speicher versorgt wird, gelangen die Befehle aus dem Speicher (also OPCODEs) in das Leitwerk. Das Leitwerk kümmert sich um die richtige Sequenz bei der Befehlsausführung und wird entsprechend dieser Aufgabe auch Sequence Unit oder Control and Timing Unit genannt. Der Control and Timing Unit ist ein Befehlsdecoder und ein Befehlsspeicher vorgeschaltet. Jeder Operationscode aus dem Speicher gelangt in den Befehlsspeicher und wird vom Befehlsdecoder auf die Funktionsweise hin untersucht. Je nach Befehl beginnt dann die Control and Timing Unit mit der schrittweisen Be-

arbeitung des vorgelegten Befehls. So werden z.B. bei einer Addition die beiden Operanden in die Quellregister geladen, dann die Addition selbst in der ALU durchgeführt und schließlich das Ergebnis vom Zielspeicher in ein Register geladen. Jeder Befehl besteht damit aus einer Kette von Einzelbefehlen, die in ihrer Summe den Grundbefehl aus dem Speicher ausmachen. Die Abfolge dieser kleinsten Einheiten bezeichnet man auch als Mikroprogramm der Zentraleinheit. Es ist bei den meisten Prozessoren fest verdrahtet. (Einige Mikrorechner lassen sich auch im Mikroprogramm von außen beeinflussen.) Jeder einzelne Mikroprogrammschritt wird in der kleinstmöglichen Zeiteinheit durchgeführt. Diese Zeiteinheit wird als Takt oder State bezeichnet. Die Dauer eines States ergibt sich bei allen Prozessoren aus einer festen Teilung der Clock-Rate oder des angeschlossenen Quarzes. Beim Atari ST wird der 68000 mit 8 MHz betrieben, was zu einer Taktlänge von 125 ns (Nanosekunden) führt. Bild 2.9 zeigt den internen Aufbau der Control and Timing Unit eines Zentralprozessors.

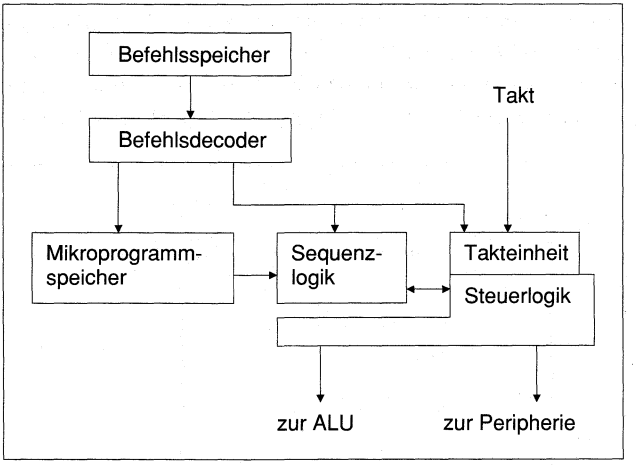


Bild 2.9: Prinzipieller Aufbau der Control and Timing Unit einer CPU

2.3 Die Hardware des Prozessors 68000

Der Mikroprozessor 68000 wurde von der Fa. Motorola entwickelt. Im Gegensatz zu dem Intel-Prozessor 8086 verfügt er über eine interne 16-Bit-Struktur, die sich bis auf 32 Bit kaskadieren läßt. Auf eine interne Akkumulatoradressierung wurde verzichtet. So lassen sich alle in der CPU befindlichen Datenregister mit gleichartigen Befehlen ansprechen. Sie können entweder mit 8, 16 oder 32 Bit verwaltet werden. Der Umfang des Adreßbusses beträgt 23 Leitungen, womit insgesamt 16 Mbyte linear adressiert werden können. Der 68000 besticht insbesondere durch einen großen Befehlsumfang, zahlreiche Adressierungsarten und eine schnelle Verarbeitungszeit, die eine Zykluszeit um 1,5 Mikrosekunden zuläßt. Er kann

in zwei verschiedenen Betriebsarten betrieben werden, im sog. Benutzermodus oder im Systemmodus. Dadurch ist eine strikte Trennung von Betriebssystem und Benutzerprogramm möglich. Zur Verwaltung der beiden Modi verfügt der 68000 über zwei Stack-Register: USP (User Stack Pointer, Benutzer-Stapel-Zeiger) und SSP (Supervisor Stack Pointer, System-Stapel-Zeiger). Beide Stack-Register erlauben eine beliebige Schachtelung von Unterprogrammaufrufen. Wie das Blockschaltbild des 68000 zeigt, verfügt er extern über einen 16 Bit breiten Datenbus (Bild 2.10). Dieser gelangt über den Datenbusempfänger in das Prefetch-Register. Dort wirkt dieses einerseits auf den internen 32 Bit breiten Datenbus, und andererseits auf die Mikrocode- und Steuerlogikeinheit. In der letzteren Einheit wird der ankommende Befehl dekodiert und interpretiert und führt folglich zu der Ausführung des Befehls selbst. Die Funktions- und Folgesteuerung, die hinter der Mikrocode- und Steuerlogik liegt, benutzt insgesamt die oben bereits erwähnten User Stack Pointer und einen Supervisor Stack Pointer sowie sieben Adreßregister mit der Bezeichnung AO bis A6. Ebenfalls befindet sich dort ein Programmzähler, der zur Dekodierung des internen Adreßbusses dient. Die interne Daten-ALU wirkt auf den Datenbustreiber und verwaltet die acht Datenregister und das Statusregister. Aus dem Datenbustreiber werden diejenigen Funktionen abgeleitet, die für die Steuerbuslogik und das Timing des Adreßbustreibers notwendig sind.

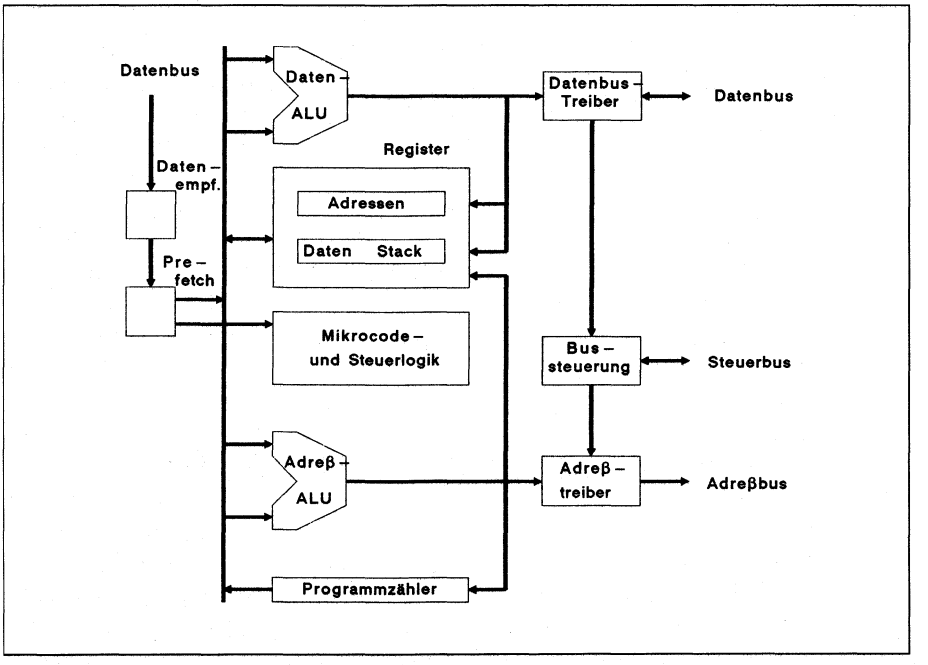


Bild 2.10: Blockschaltbild des Prozessors 68000

Der 68000 versteht 56 Grundbefehle, die insgesamt 14 Adressierungsarten zulassen (es sind aber nicht alle Adressierungsarten bei jedem Befehl erlaubt). Bei den Befehlen handelt es sich um Transportbefehle, arithmetische und logische Befehle, absolute und relative Sprungbefehle, Vergleichs-, Schiebe und Spezialbefehle. Der 68000 ist zur Zeit der wichtigste Prozessor aus der Mikroprozessorfamilie 680xx von Motorola, zu dem noch die Prozessoren 68008, 68010, 68020 und 68030 gehören. Dabei handelt es sich bei dem 68008 um eine kompatible Version des 68000, wobei der Datenbus mit einer 8-Daten-Bit-Breite ausgeführt ist. Der 68010 enthält zusätzliche Befehle und erlaubt eine Verarbeitung in einer virtuellen Speicherverwaltung. Die Prozessoren 68020 und 68030 sind im Gegensatz zum 68000 32-Bit-Versionen für eine extrem schnelle Datenverwaltung. Alle Prozessoren sind zum 68000 aufwärtskompatibel.

2.4 Kundenspezifische Integration

Die kundenspezifische Integration von integrierten Schaltungen gibt es fast so lange wie die Integration von Standardschaltungen. Prinzipiell wurde diese Entwicklung durch die Erfindung des Planartransistors ermöglicht. Aber erst durch die Weiterentwicklung der CMOS-Technologie gelang es, eine derart hohe Integrationsdichte zu erreichen, wodurch man auch recht komplexe Funktionen in einem einzelnen Baustein ohne thermische Überlastung unterbringen konnte. Anfangs lohnten sich derartige Entwicklungen nur bei einer Abnahme von mehr als 100 000 Bausteinen pro Jahr. Im Laufe der Zeit wurde die kundenspezifische Integration immer billiger und heute lohnt sich eine solche Integration bereits ab ca. 1000 Bausteinen pro Jahr. Beim sogenannten Full Custom Design wird jedes benötigte Element, sei es ein Transistor, ein Widerstand oder eine Verbindungsleitung, entsprechend der benötigten Spezifikation speziell zugeschnitten. Die daraus entstehende integrierte Schaltung zeigt unter den gegebenen Möglichkeiten des eingesetzten Halbleiterprozessors die bestmöglichen Werte für Temperaturverhalten, Treiberfähigkeit und Verarbeitungsgeschwindigkeit. Dabei können sowohl analoge als auch digitale Funktionen auf ein und demselben Chip implementiert werden. Die Entwicklungsaufwendungen sind jedoch in den meisten Fällen sehr hoch. Fast jede Schaltung muß mit hohem Zeitaufwand und extrem komplexen Layout angefertigt werden. Daher geht man bei der heutigen Entwicklung hochintegrierter Schaltelemente häufig dazu über, fertige Funktionselemente zu benutzen, wozu z.B. Operationsverstärker, Spannungs- und Referenzquellen, Addierer, Zähler, logische Gatter und ähnliche Elemente gehören. Jedes dieser Funktionselemente bildet auf dem späteren Chip eine Zelle. Jede dieser Zellen wird mit gemessenen Parametern und einem Simulationsmodell der realisierten Funktion getestet. Alle so erhaltenen Informationen werden zusammen mit Layout der Zelle in einer Bibliothek auf dem Entwicklungs-Rechner abgelegt. Man kann so auf eine große Zahl von ausgetesteten Funktionen zurückgreifen und vermeidet in der Regel zahlreiche Redesigns (Neugestaltung des Chip-Layouts). Dieses Verfahren liegt im Preis daher erheblich niedriger als ein Full Custom Design.

Der nächste Schritt in der Integrationstechnik bestand nun darin, Chips anzubieten, die alle benötigten Funktionselemente bereits enthalten und nur noch richtig konfiguriert werden müssen. Wie in Kapitel 1 erläutert, können alle Schaltungen aus nur wenigen logischen Funktionen aufgebaut werden. So enthalten die sogenannten Gate-Arrays (Gatter-Raster) die grundlegenden logische Elemente, die innerhalb der Chips nur noch richtig verdrahtet werden müssen. Die Entwicklung für eine spezielle Anwendung reduziert sich bei Gate-Arrays auf das Erstellen von Masken, die eine richtige Zuordnung (Verdrahtung) der Hardwarezellen durchführen. Die Integration von Gate-Arrays erlaubt es heute, einige Tausend Funktionsgatter miteinander zu verbinden. Mittlerweile können neben digitalen Funktionen auch analoge Funktionen direkt integriert werden.

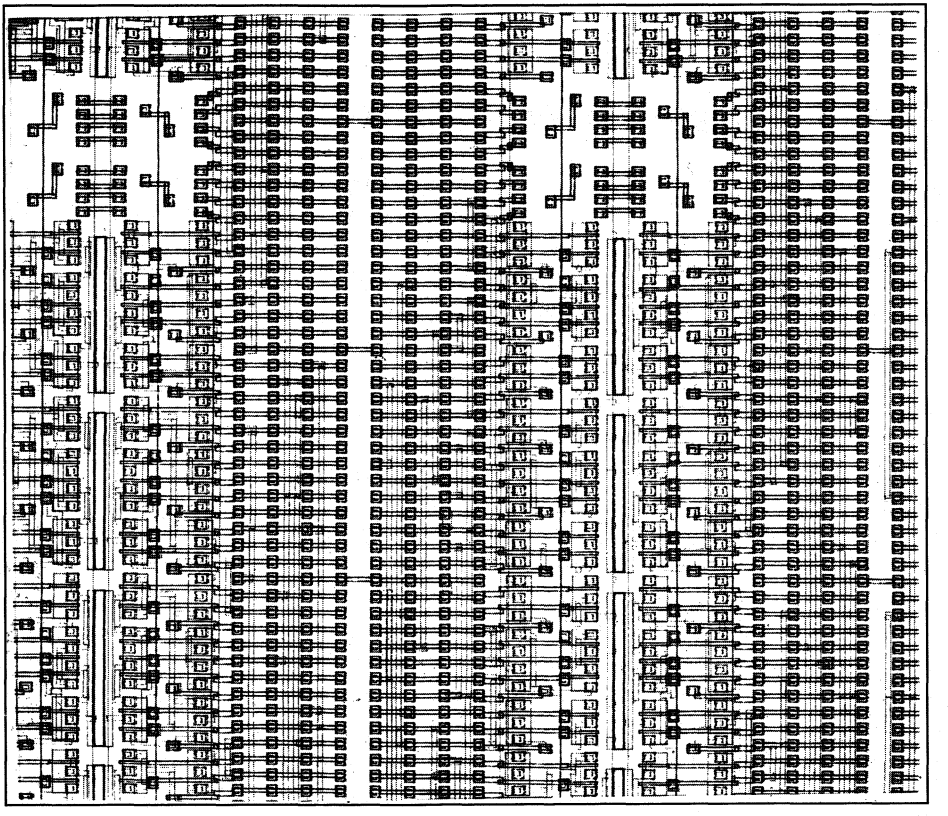


Bild 2.11: Aufbau einer Standardzelle (Gate-Array)

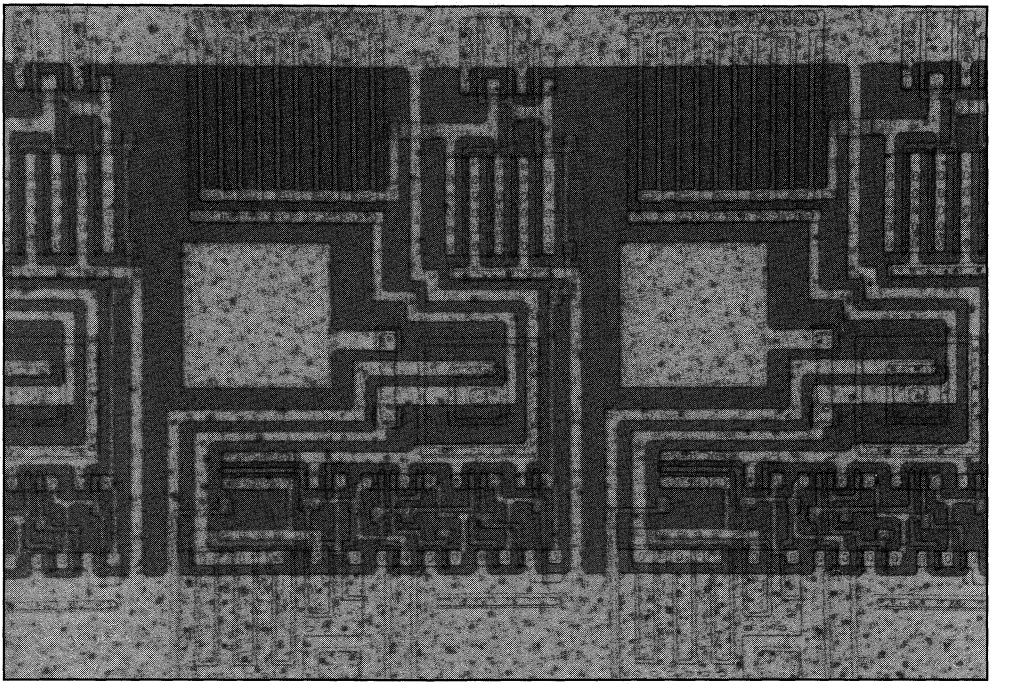


Bild 2.12: Darstellung einer Verdrahtung in einem Gate-Array

Beim derzeitigen Stand der Technik wird ein 2-CMOS-Prozeß verwendet, der in Zukunft entweder durch einen 1,5- oder 1,0-Prozeß ersetzt werden wird. Bild 2.11 zeigt die Vergrößerung einer Standardzelle, wie sie bei der Integration von Gate-Arrays Verwendung findet. In dem abgebildeten Beispiel handelt es ich um eine Treiberzelle. Die typische Verdrahtung eines Gate-Arrays ist im Bild 2.12 zu sehen.

Im Atari ST befinden sich mehrere hochintegrierte Bausteine, die durch eines der in diesem Kapitel vorgestellten Verfahren hergestellt wurden. Nur so war der Aufbau auf kleinstem Raum bei äußerst geringem Preis überhaupt möglich.

3 Datentransport

3.1 Kommunikationswege und Datentransportstrukturen

Erst durch Kommunikation mit der Außenwelt wird ein Rechner zum Werkzeug für anwendungsorientierte Problemlösungen. Ohne Datentransport bleibt der Rechner nur ein Mittel zum Selbstzweck. Es ist dabei gleichgültig, ob es sich bei den Daten um Inhalte von Speicherplätzen, um Meßwerte oder Druckeranforderungen handelt. Daten sind Informationen, die über unterschiedliche Entfernungen mit unterschiedlichen Geschwindigkeiten transportiert werden müssen. Je nach Anwendung haben sich deshalb in den letzten 30 Jahren verschiedene Typen von Datenverbindungen durchgesetzt, die auf die jeweiligen Anforderungen zugeschnitten sind.

Die Struktur der Datenverbindung hängt nicht davon ab, ob die Daten innerhalb oder außerhalb des Rechners transportiert werden, vielmehr wird die Struktur von der Datenrate, von den Umweltbedingungen, von der Geschwindigkeit und von dem Zugriffsverfahren bestimmt. Topologisch sind drei Transportverfahren üblich: sternförmige Verbindungen, Ringverbindungen und Bussysteme (Bild 3.1).

Die geometrische Anordnung alleine gibt noch keinen Aufschluß über den Transportmechanismus, sondern läßt lediglich Schlüsse über die grobe Struktur der elektrischen Leitungen zu. Neben der angewandten Topologie sind Kenntnisse über die festgelegten Normen notwendig, die eine Datentransportprozedur vollständig beschreiben. Zu diesen Konventionen gehören die Angaben über Spannungspegel, Stromverlauf, Kabeleigenschaften (also die elektrischen Werte) und die Angaben über das Transportverfahren (also die Prozeduren der Datenformatierung). Beide Arten von Konventionen lassen praktisch unendlich viele Möglichkeiten des Datentransports zu und können auf jede Anwendung zugeschnitten werden. Bei einem Problem mit Datenübertragung empfiehlt es sich deshalb, erst die Normen zu studieren und dann eine geeignete Konvention zu wählen. Eigene Wege sind in der Regel mit doppelter Arbeit verbunden und erschweren jedes Streben nach Kompatiblität.

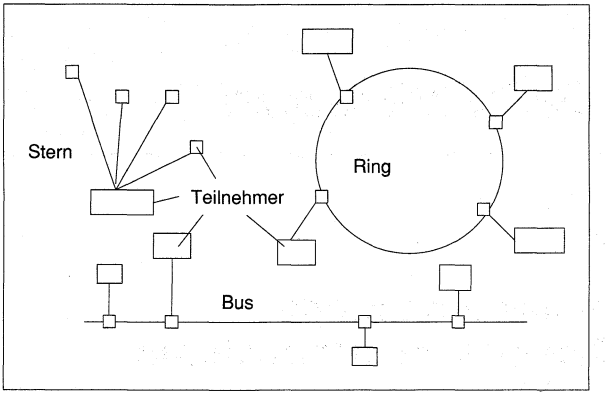


Bild 3.1: Die drei wichtigsten Datentransportverfahren: sternförmige Verbindungen, ringförmige Verbindungen und Busverbindungen

Die einfachste Art der Datenkommunikation besteht in einer sternförmigen Verbindung. Dabei ist jeder Kommunikationstrecke ein eigenes Übertragungssystem zugeordnet, das nur für die spezielle Aufgabe zur Verfügung steht. Sternförmige Verbindungen werden auch Punkt-zu-Punkt-Verbindungen genannt und sind beim Anschluß von Druckern, Datensichtgeräten oder Plottern sinnvoll. Der Datentransport bei einem Bussystem ist innerhalb des Rechners die am weitesten verbreitete Methode. Hierbei werden Daten von einem bestimmten Sender zu einem Empfänger transportiert. Die Transportleitung wird jedoch nicht – wie bei der sternförmigen Verbindung – nur von einem Sender und einem Empfänger benutzt, sondern bietet einer großen Anzahl von Teilnehmern gleichartige Zugriffsmöglichkeiten. Ringförmige Strukturen bestehen aus Punkt-zu-Punkt-Verbindungen, die jeweils einen Sender mit einem Empfänger verbinden. Der Empfänger enthält einen internen Sender, der die Daten entweder direkt vom Empfänger weitergibt oder nach einer Umsetzung erneut auf die Reise schickt.

Innerhalb der Hardware kennt man zwei unterschiedliche Verfahren, die Daten dem Transportmedium anzubieten. Parallele Systeme haben eine feste Anzahl von Bits (entweder als Byte oder Wort), die gleichzeitig auf mehreren Leitungen transportiert werden. Bei seriellen Übertragungsarten wird eine Information in zerhackter Form zeitlich hintereinander auf einer Leitung weitergeben. Da sich ringförmige Strukturen bis heute nur bei lokalen Netzen (LANs) durchgesetzt haben, beschränkt sich die folgende Beschreibung auf parallele und serielle Transportmechanismen in stern- und busförmigen Systemen.

3.2 Transportmedien

Die Anwendung bestimmt nicht nur die Topologie der Datenstrecke, sondern auch das Transportmedium. Heute sind elektrische Verbindungen am weitesten verbreitet. Neben dieser Transportart sind u. a. auch Funkverbindungen und optische Verbindungsstrecken gebräuchlich. Gerade optische Verbindungen lösen heute an vielen Stellen die elektrischen ab, da optische Transportmedien keinerlei Probleme beim Potentialausgleich haben und sich durch eine geringe Störanfälligkeit auszeichnen.

3.3 Parallele Bussysteme innerhalb des Rechners

Parallele Bussysteme werden überall dort eingesetzt, wo auf einem eng begrenzten Raum möglichst schnell Daten ausgetauscht werden sollen. Innerhalb eines Rechnersystems erfolgt der Datentransport zwischen der CPU und dem Speicher oder der Peripherie über parallele Busstrukturen. Bei parallelen Bussystemen wird eine Information, die aus 8 oder 16 Bit besteht, auf einen Schlag übermittelt. Da dieser Vorteil mit einem hohen Leitungsaufwand bezahlt werden muß, kommen Bussysteme normalerweise nur bei geringer Ausdehnung und bei schnellem Datenverkehr zum Einsatz. Die Philosophie der Bussysteme ist weit verbreitet. Selbst Speicherbausteine oder Peripheriekomponenten beinhalten interne Bussysteme, die eine Erweiterung der externen darstellen.

Parallele Bussysteme bestehen in der Regel aus drei Unterbereichen, die jeweils wieder einen eigenen Bus darstellen. Der erste Bereich ist der Adreßbus, der vom jeweiligen Sender erzeugt wird und für die Anwahl des Empfängers zuständig ist. Die zweite Komponente ist der Kontrollbus, der über die Aktivität und die Datentransportrichtung Auskunft gibt. Das dritte Element ist der Datenbus, der die Parameter und Daten der Übertragung enthält. Der Transport von Daten auf einem derartigen System ist überaus einfach: Zuerst gibt der Sender eine Adresse über den Adreßbus aus und selektiert damit den gewünschten Empfänger. Als Empfänger eignen sich Speicherelemente, Peripheriebausteine, eventuell fremde CPUs usw. Nach dem Anlegen der Adresse erscheinen die Daten zuerst am Datenbus und werden dann über die Schreib-Aufforderung (WRITE) zum Empfänger gebracht. Möchte die CPU ihrerseits Daten lesen, so sendet sie ebenfalls die Adresse aus und aktiviert den Empfänger durch das Kontrollbussignal Lesen (READ). Dadurch bringt der Empfänger seine Dateninformation auf den Datenbus und wird damit selbst zum Datensender. Bild 3.2 zeigt den Ablauf dieser beiden Prozeduren in einem Timing-Diagramm.

Bussysteme, die in der beschriebenen Art und Weise arbeiten, sind in allen Rechnertypen üblich. Unterschiede existieren lediglich in der Datenbusbreite, in der Funktionsweise des Kontrollbusses und in den logischen Datenpegeln. In vielen Fällen werden auch Bussysteme benutzt, die über einen gemeinsamen Daten-Adreßbus verfügen. Solche Bussysteme benöti-

gen wesentlich weniger Leitungen, brauchen aber jeweils eine Kontrolleitung, die entscheidet, ob es sich bei der Busübertragung um eine Adreß- oder Dateninformation handelt. Für Rechnertypen der Von-Neumann-Struktur ist ein derartiges Bussystem absolut hinreichend. Es erlaubt das Adressieren externer Speicher oder E/A-Bereiche, es ermöglicht den Daten- und Parametertransport und es beinhaltet die notwendigen Kontrollsignale zur Vorgabe der Datentransportrichtung.

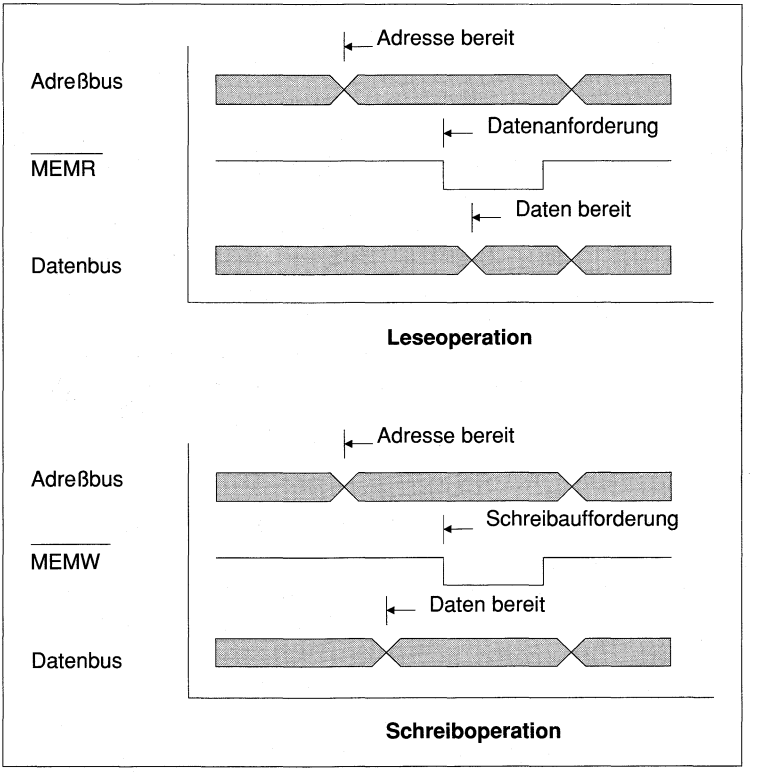


Bild 3.2: Zeitliches Zusammenwirken der Adreß-, Daten- und Kontrollbussignale

Bild 3.3 zeigt, welche Transportprozeduren am Bus abgewickelt werden, damit eine Information von der Peripherie in die CPU gelangt. Zuerst adressiert die Zentraleinheit den Speicher (1), um den Operationscode zu holen. Dieser Operationscode gelangt durch die Aufforderung MEMR über den Datenbus in die CPU und wird dort interpretiert (2). Hinter dem Opcode verbirgt sich ein Befehl, der eine Information von der Peripherie holen soll. Dazu adressiert die Zentraleinheit im nächsten Schritt die entsprechende Peripherieeinheit (3) und fordert die Information über diesen Adreßbereich an. Der gesamte Befehl wird abgeschlossen, indem der Zustand der Peripherie – als Datum interpretiert – zur CPU gebracht wird (4).

Als weitere Kontrollmöglichkeit verfügen viele Bussysteme über sogenannte Interruptleitungen. Signale auf diesen Leitungen befähigen die Peripheriekarten, die Zentraleinheit auf ihre Bedürfnisse einzustellen. So kann eine Peripherieeinheit über diese Leitungen eine Programmunterbrechung anfordern und die CPU auf ein besonderes Ereignis hinweisen (siehe Kapitel 5).

Der ROM-Port des Atari ST stellt nur einen Teil des tatsächlichen internen Bussystems dar. Bild 3.4 zeigt die Anschlußanordnung mit den Bereichen für eine externe Versorgung, den Daten- und den Adreßleitungen. Die Leitung CS (ROM-PORT) mit der Nr. 33 selektiert den Lesezugriff über die angeschlossenen Einheiten.

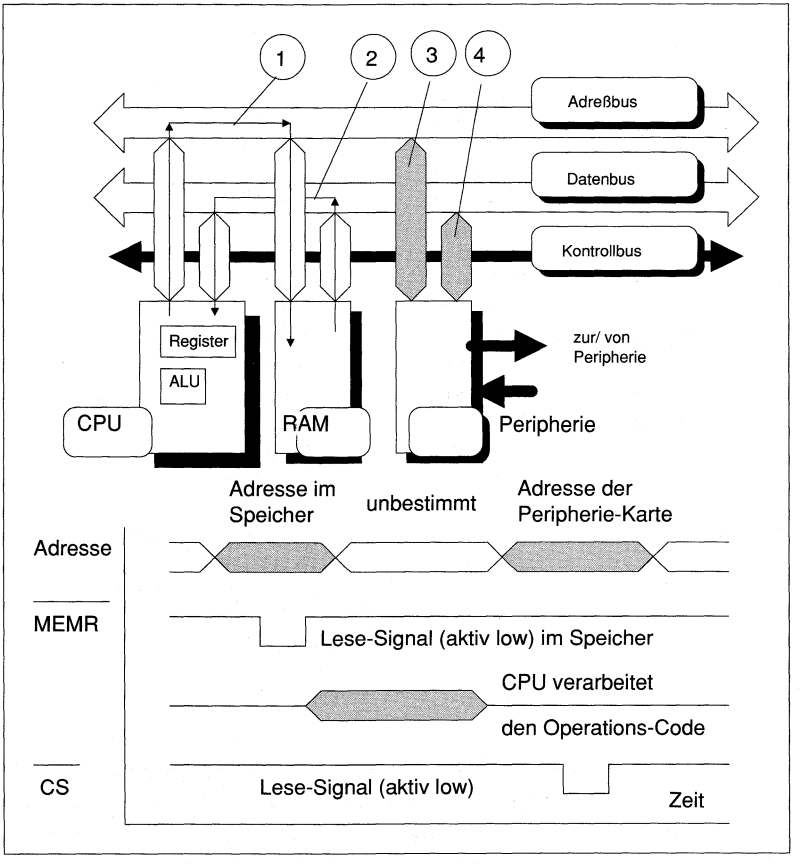


Bild 3.3: Prinzipieller Aufbau eines Computersystems und das zeitliche Verhalten einiger Busleitungen

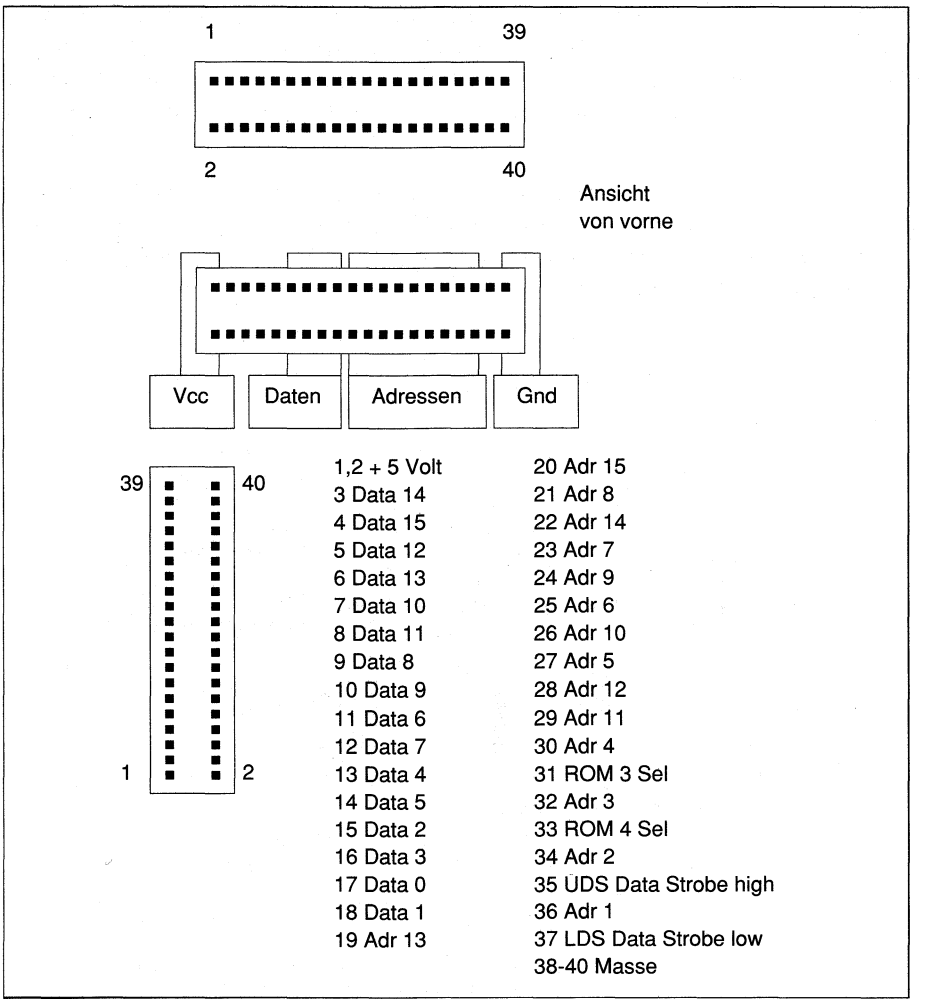


Bild 3.4: ROM-Port des Atari ST

Neben dem Bussystem des Atari ST sind zahlreiche Parallelbussysteme gebräuchlich. Auf der Basis des Z80 (8080,8085) existiert ein weit verbreitetes System unter dem Namen ECB-Bus. Dieses Bussystem sieht die Anordnung im Europakartenformat (100 mm x 160 mm) vor. Der VMEbus (Versa Module Europa) basiert auf Rechnertypen aus der Motorola-Serie (z.B. 68000) und verwendet ein Kartenformat nach den DIN-Normen 41612 und 41494. Der VMEbus enthält 16 Daten- und 24 Adreßleitungen. Auf den Prozessoren der Firma Intel basiert der Multibus (IEEE 796), der über den gleichen Daten- und Adreßbereich wie der

VMEbus verfügt. Im Gegensatz zum VMEbus beinhaltet der Multibus jedoch getrennte Bereiche zwischen der Speicher- und E/A-Adressierung.

In den USA trifft man das S100-Bus (IEEE 696) oftmals an. Er basiert auf dem Prozessor des Typs 8080 und sieht ein Kartenformat mit den Maßen 254 mm x 127 mm vor. Weit verbreitet ist auch das Bussystem der IBM-Personalcomputer. Es verfügt über getrennte Daten- und Adreßbusleitungen und beinhaltet neben zahlreichen Kontrollbussignalen auch Anschlüsse zur Versorgung der Interfacekarten. Die Liste der Bussysteme läßt sich fast beliebig fortführen, da nahezu jeder Hersteller von Prozessoren seine eigene CPU-Architektur im Bussystem wiederfinden will und damit ein spezielles Konzept unterstützt. Bild 3.5 zeigt den Aufbau eines typischen Bussystems für Karten im Europakartenformat.

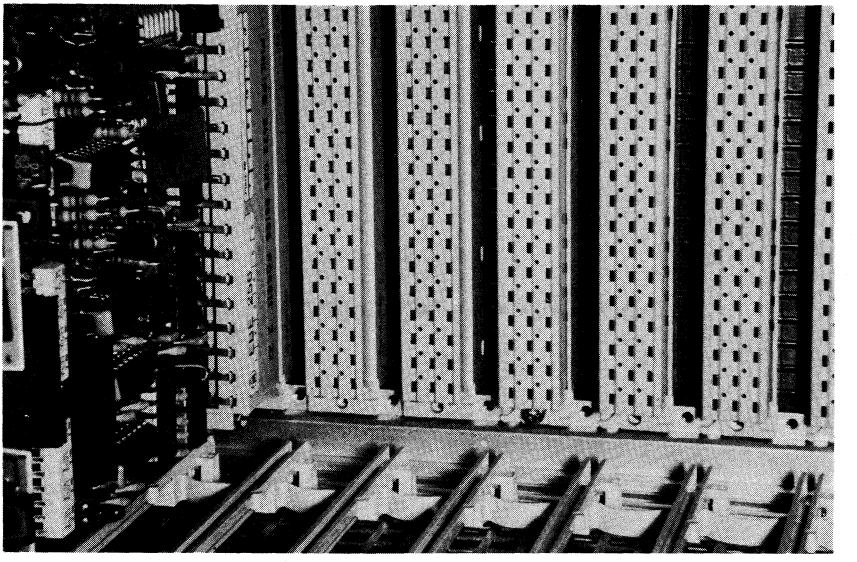


Bild 3.5: Bussystem bei einer »speicherprogrammierbaren Steuerung« der Firma Klöckner-Moeller

3.4 Parallele Bussysteme außerhalb des Rechners

Periphere Bussysteme unterscheiden sich strukturell kaum von lokalen Systemen innerhalb einer Rechnereinheit. Sie verfügen jedoch zusätzlich über spezielle Hardware-Eigenschaften und Software-Prozeduren, die den etwas rauheren Umweltbedingungen außerhalb des Gehäuses angepaßt sind. Beispielsweise verfügen die Busteilnehmer über Leistungstreiber, die durch höhere Steuerströme auch längere Leitungsdistanzen überbrücken können. Die Schalt-

pegel, welche den logischen Zuständen entsprechen, beinhalten in der Regel eine größere Hysterese, so daß eventuelle Störungen und ein eingekoppeltes Rauschen nicht zum Tragen kommen. Softwareseitig fügt man der reinen Dateninformation spezielle Zusatzdaten (Redundanz) hinzu, mit denen sich eventuelle Fehler leicht erkennen lassen.

Das wohl bekannteste periphere Bussystem ist das IEEE-488-Bussystem (auch bekannt unter den Namen IEC oder HP-IB). Dieses Bussystem dient der Ansteuerung und dem Datenaustausch zwischen Peripherieeinheiten und einem Rechner. Zu diesen externen Geräten zählen z.B. Digitalvoltmeter, Taktgeneratoren, Drucker, Massenspeicher, Logikanalysatoren, Netzgeräte usw. Insgesamt vermag das IEEE-488-System bis zu 15 unterschiedliche Geräte miteinander zu verbinden und Daten, Parameter und Steueranweisungen auszutauschen. Jedes Gerät nimmt in einer speziell auf die Funktionsweise abgestimmten Art am Busgeschehen teil. So kann der Teilnehmer des IEEE-488-Bus entweder als Commander, Listener oder Talker fungieren.

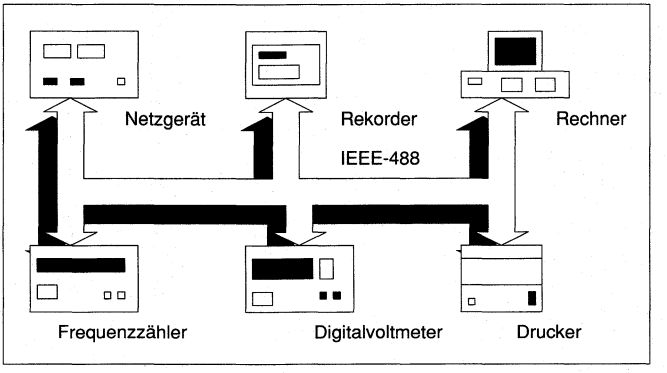


Bild 3.6a: Beispiel für den IEEE-488-Bus. Unterschiedliche Teilnehmer werden durch einen parallelen Bus verbunden

Die Commanderfunktion kommt in der Regel am Bus nur einmal vor und wird meistens durch den Zentralrechner realisiert. Der Commander führt alle Funktionen am Bus aus und entscheidet darüber, in welcher Form der Datentransport abläuft. Listener sind diejenigen Geräte, die nur Daten empfangen (z.B. ein Drucker oder ein Generator). Als Talker arbeiten alle Geräte, die auf Anfragen mit einer Datenübertragung reagieren (z.B. Digitalvoltmeter oder Pegelmeßgeräte). Darüber hinaus gibt es noch eine Klasse von Geräten, die mehrere Funktionen am Bus erfüllen. So kann z.B. ein Logikanalysator auch Listener (Bereichseinstellung) und Talker (Meßwerterstellung) sein. Auch der Commander beinhaltet häufig noch zusätzliche Funktionen. Bild 3.6a gibt einem Einblick in die Vielfalt der Anschlußmöglichkeiten beim IEEE-488-Bus. Bild 3.6b zeigt die Steckerbelegung des IEEE-488-Interfaces.

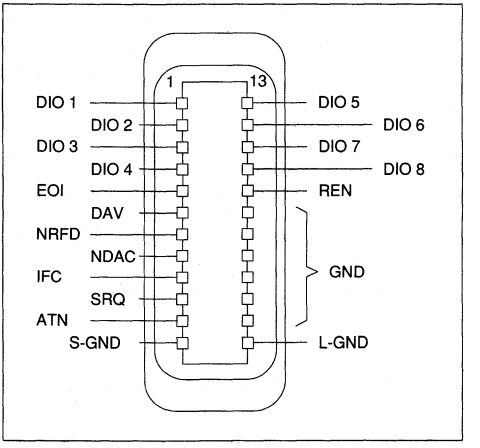


Bild 3.6b: Der Steckverbinder des IEEE-488-Bussystems

Neben den reinen Dateninformationen existieren auf dem IEEE-488-Bus noch acht weitere Leitungen, die für den Datentransport und die richtige Synchronisation zuständig sind. Der Datentransport geschieht in allen Fällen mit Handshake, das heißt, jeder Datentransport wird vom Empfänger quittiert. Um ein Datum zu transportieren, legt der Sender zuerst sein Datum auf die Datenleitungen des Bussystems und wartet, bis alle Teilnehmer den Zustand NRFD (Not Ready For Dates) aufgehoben haben.

Erst wenn der letzte Teilnehmer seine Empfangsbereitschaft auf diese Art angezeigt hat, erzeugt der Sender ein DAV-Signal (Data Valid). Jeder Teilnehmer nimmt nun dieses Datum entgegen und verarbeitet es intern. Solange das Datum noch nicht eingelesen ist, signalisiert der Empfänger dem Sender durch das NDAC (No Data Accepted), das Datum noch beizubehalten. Erst wenn der letzte Teilnehmer das Datum verstanden hat, hebt der Sender das DAV-Signal auf und entfernt das Datum vom Bus (Bild 3.7).

Die Datenleitungen dienen beim IEEE-488-Bus auch zur Adressierung. Möchte der Sender (Commander) eine spezielle Einheit neu adressieren, so sendet er während der Adreßinformation das Signal ATN (Attention). Die angeschlossenen peripheren Einheiten reagieren folgendermaßen auf ATN: Das Datum wird eingelesen und mit ihrer intern gespeicherten Adresse verglichen. Sofern die Adreßwahl nicht der peripheren Einheit entspricht, bleiben sie weiter inaktiv. Sollte die Adresse jedoch identisch zu der Geräteadresse sein, so nehmen sie an dem weiteren Busgeschehen teil. Die folgende Tabelle beinhaltet die Bezeichnungen des IEEE-488-Bussystems.

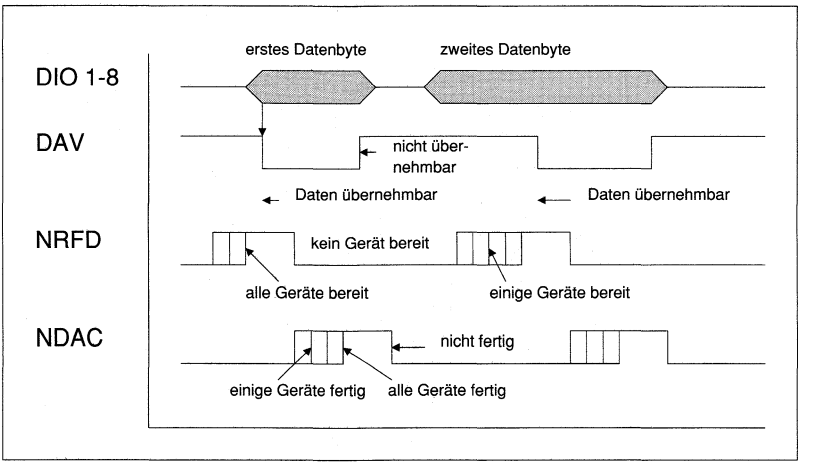


Bild 3.7: Timing-Diagramm des IEEE-488-Busses

Name	Funktion	Kontakt
Datenleitungen:		
DIO	Datenein- und -ausgabe	1 – 4,14 – 17
Handshakeleitungen:		
DAV	Data Valid, Daten gültig	7
NRFD	Not Ready For Data, (–) keine Bereitschaft für Daten	8
NDAC	Not Data Accepted, (–) Daten noch nicht übernommen	9
Managementleitungen:		
ATN	Attention, Achtung	12
SRQ	Service Request, Aufruf	11
EOI	End or Identify, Ende oder Identifizierung	6
REN	Remote Enable, Fernsteuerfreigabe	5
IFC	Interface Clear, Funktionen rücksetzen	10

Die IEEE-488 stellt nicht nur eine elektrische Schnittstelle dar, sondern trifft auch Vereinbarungen über das Protokoll. Dabei haben nicht nur die ausgetauschten Daten eine festgelegte Bedeutung, sondern alle Transportprozeduren sind nach Diensten definiert. Zu jedem Dienst

gehört eine komplette Übertragungsprozedur, die aus jeweils einer größeren Anzahl an Bytes besteht. Innerhalb der Norm können einige Geräte nur Teilfunktionen (Untermenge der Dienste) unterstützen, sofern sie durch ihre Hardware eine bereits eingeschränkte Funktionalität haben. So braucht ein Drucker nur die Dienste eines Hörers (Listener) beinhalten; Commander-Funktionen sind für ihn überflüssig.

Je komplexer die Beschaltung einer Schnittstelle wird und je mehr Funktionen vom Datentransport gefordert werden, desto häufiger findet man eine genaue Abgrenzung zwischen den einzelnen Funktionsgruppen beim Datentransport. Eines der am weitesten verbreiteten Modelle ist das ISO (International Standards Organisation) 7-Schichten-Modell, bei dem jede Schicht ihre festgelegte Aufgabe beim Datenverkehr wahrnimmt. Die unterste Schicht (Bild 3.8) stellt die physikalische Ebene dar, in der alle Konventionen über das Transportmedium und Transportverfahren getroffen werden. Die oberste Schicht korrespondiert mit der speziellen Anwendung und kann als Benutzeroberfläche bezeichnet werden. Bei der Kommunikation zweier Benutzer sieht es deshalb so aus, als ob sich jede Schicht nur mit der entsprechenden Schicht seines Gegenübers unterhält. So verständigen sich die physikalischen Schichten nur über das Medium und bereiten die Bitinformationen auf, ohne auf deren Bedeutung einzugehen. Die Anwenderschichten sehen nur die Dienste und den Informationsaustausch und brauchen sich nicht um den Datenverkehr oder die Datensicherung zu kümmern. Die einzelnen Schichten des ISO-Modells haben folgende Bedeutung:

1. Schicht
Physikalische Schicht
Übertragung des Stroms zur Übermittlung der binären Information
Wahl des Leitungsmediums (Koax-, Zweidraht-, oder Lichtleiter)

2. Schicht
Verbindungsschicht
Aufbereitung der Daten, Datensicherung

3. Schicht
Netzwerk-Schicht
Übermittlungsverfahren, Wahl der Datenroute

4. Schicht
Transportschicht
Optimierung der Datenübertragung

5. Schicht
Sessions-Schicht
Koordination der Zusammenarbeit der kommunizierenden Anwendungen

6. Schicht
Präsentations-Schicht
Formatierung der Darstellung der Daten

7. Schicht
Applikations-Schicht
Festlegung der Natur der Daten, Benutzeranforderungen

Je nach Anwendungsbereich können innerhalb einer Datenübermittlung auch einige Schichten entfallen. Ein Großteil der Datenübertragungen sieht nur die oberste (7) und die unterste Schicht (1) vor. Der Programmierer kennt in der Regel bei zahlreichen Programmiersprachen nur die Aufrufe bei der Datenübertragung (z.B. eine PRINT-Anweisung) und braucht sich über die Bereitstellung des Telegramms nicht weiter zu kümmern. Für die unterste Schicht

fühlt sich lediglich ein spezieller Hardwarebaustein zuständig, der die richtige Bitzuordnung übernimmt und die Daten in der gewünschten Form der Ausgabe (z.B. Drucker) zuweist.

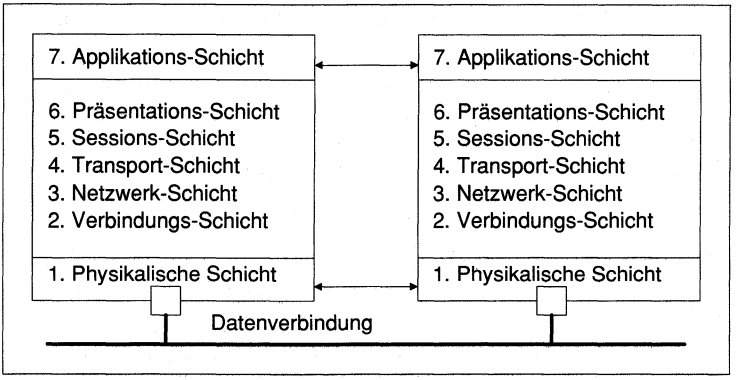


Bild 3.8: Das ISO-Schichten-Modell

In ähnlicher Form erfolgt auch die Übertragung der Daten innerhalb des IEEE-488-Systems. Der Anwender programmiert in seiner Programmiersprache nur den Funktionsaufruf (z.B. Bereichsumschaltung auf 20 mV), während die IEEE-488-Schnittstelle zur Realisierung dieser Funktion neun Zeichen zu dem gewählten Gerät transportiert. Darüber hinaus erfolgt auch das gesamte Handshake ohne Einfluß des Programmierers.

3.5 Buszugriffsverfahren

Wie in den beiden letzten Abschnitten beschrieben, beteiligen sich immer mehrere Teilnehmer am Busgeschehen. Bei einem Datenaustausch sendet der jeweilige Sender eine Information an den momentanen Empfänger. Der Sender bestimmt damit den Zustand am Datenbus und legt die entsprechenden logischen Pegel für die Übertragungsdauer fest. Der Sender benötigt dazu einen Bustreiber, der mit einem relativ hohen Strom die Busleitungen versorgt. Er muß dabei nicht nur die Empfänger treiben, sondern sich auch gegen die parasitären Kapazitäten der Busleitungen zur Wehr setzen. Treiberströme bis 48 mA sind deshalb keine Seltenheit. Je leistungsfähiger der Bustreiber ist, desto weniger Zeit braucht er, die Kapazitäten zu laden (oder entladen), und desto höher kann die Datentransferrate sein. Die Empfänger sollten ihrerseits möglichst wenig Strom von den Datensignalen aufbrauchen, so daß der Sender eine möglichst hohe Zahl an Teilnehmern treiben kann. Je nach verwendeter Logik sind Restströme von 0,2 bis 1,6 mA bei den Empfängern möglich. Die parasitären Kapazitäten betragen bei den lokalen Bussystemen zwischen 5 und 30 pF.

Die Teilnehmer, die am Busgeschehen gerade nicht teilnehmen, müssen einen hochohmigen Zustand annehmen. Im Prinzip ist es zwar auch möglich, daß ein inaktiver Teilnehmer in Empfangsposition verharrt, dieser Zustand belastet aber in fast allen Fällen den Bustransport zu stark. Moderne Bustreiber erlauben deshalb die Wahl dreier Zustände. Sie können Sender, Empfänger oder hochohmige Elemente sein. Als hochohmige Elemente ist ihrem Zustand weder eine logische 1 noch eine logische 0 zuzuordnen. Dieser Zustand wird deshalb auch als Tri-State bezeichnet. Die Busbelastung geht in diesem Zustand auf wenige Mikroampere zurück. Nahezu alle lokalen Bussysteme enthalten somit jeweils einen Sender, einen Empfänger und eine recht große Anzahl von Teilnehmern im Tri-State-Zustand. Da Sender und Empfänger in schneller Folge wechseln, müssen auch die Bustreiber laufend ihre Transportrichtung und ihre Zustände ändern. Bild 3.9 zeigt ein typisches Datenbussystem mit vier angeschlossenen Teilnehmern.

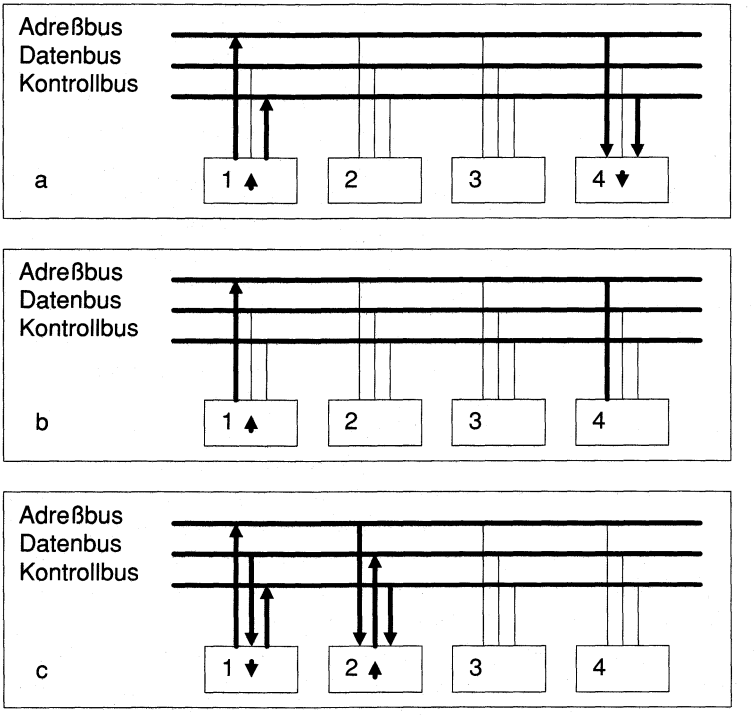


Bild 3.9: Einheiten tauschen Informationen über den Bus aus

Die Einheit links fungiert als momentaner Sender und betreibt den Bus. Die rechte Einheit ist der Empfänger, mit dem der Sender Kontakt aufgenommen hat. Über den Adreßbus wurde diese Einheit bereits aktiviert. Der Kontrollbus (WRITE) hat die Transportrichtung des Bus-

treibers der rechten Einheit nach innen orientiert (a). Die beiden mittleren Einheiten reagieren nicht auf die Adresse der rechten Einheit und verbleiben hochohmig (Tri-State). Im nächsten Schritt (b) hat der Sender seine Operation abgeschlossen und deaktiviert die WRITE-Funktion. Die rechte Einheit wird damit auch hochohmig. Im letzten Schritt (c) fordert die linke Einheit eine Information von der zweiten Einheit an. Dazu sendet sie die Adresse der zweiten Einheit aus und aktiviert den Kontrollbus (READ). Die zweite Einheit wird damit zum Sender, während die erste Einheit den Bustreiber nach innen schaltet und die angeforderte Information erhält.

Der IEEE-488-Bus kennt die Tri-State-Funktion nicht. Er benutzt ein anderes Verfahren, das unter Open-Collector bekannt ist. Dazu verfügen alle Sender über eine Ausgangsstufe, die den angeschlossenen Bus nur nach logisch 0 zu ziehen vermag. Eine logische 1 entspricht gleichzeitig dem nicht aktiven Zustand. Der externe Bus enthält seinerseits an einem Teilnehmer ein Widerstandsnetzwerk, das die Busleitungen nach oben (also gegen + 5 Volt) zieht. Dadurch gibt es nur einen aktiven Pegel, wenn ein oder mehrere Teilnehmer die Busleitung nach unten ziehen. Durch dieses Verfahren können mehrere Benutzer ihren Zustand über eine Leitung mitteilen. So erfährt der jeweilige Sender nicht im einzelnen, welche Einheit die Daten akzeptiert hat und welche noch nicht bereit ist; lediglich wenn die letzte Einheit die Daten aufgenommen hat, wird der externe Widerstand wirksam und die logische Funktion wechselt von 0 auf 1. Technisch ist eine derartige Schaltweise durch einen Transistor in der Ausgangsstufe zu realisieren, dessen offener Kollektor die Busleitung betreibt.

Die schnelle Abfolge von unterschiedlichen Zuständen am Bus bringt bei kurzen Verarbeitungszeiten große Probleme mit sich. Beim Aufbau werden logische Familien verwendet, die nicht beliebig schnell schalten können. Interne Laufzeiten der verwendeten Gatter oder der Dekoder tragen ganz erheblich zu Zeitdifferenzen bei. Eine nicht seltene Verknüpfung von vier bis fünf Elementen zum Realisieren der logischen Funktion erfordert Durchlaufzeiten bis 60 ns und mehr. Es kann deshalb zu fatalen Fehlern auf dem Bus kommen, wenn sich eine Einheit von der Busaktivität noch nicht zurückgezogen hat, während bereits eine andere den Bus treibt. Alle Zugriffszeiten müssen daher gut aufeinander abgestimmt sein, damit es auch im schlimmsten Fall (Worst-Case) zu keinerlei Kollisionen kommt. Beim Design eines Bussystems spielt deshalb nicht nur die logische Funktion eine wichtige Rolle, es muß ein Laufzeitdiagramm erstellt werden, in dem die Grenzwerte (Worst-Case) verzeichnet sind. Bild 3.10 zeigt einen realen Zugriff der Zentraleinheit mit einer Leseanforderung an die Peripherie. Das Timing entspricht den tatsächlichen Verhalten beim Lesezugriff des Atari ST auf den ROM-Port. Ungefähr 50 ns nach dem Einschwingen der Adressen signalisiert die CS-Leitung den Peripheriezugriff. Die Daten müssen dann innerhalb von 280 ns zur Verfügung gestellt werden.

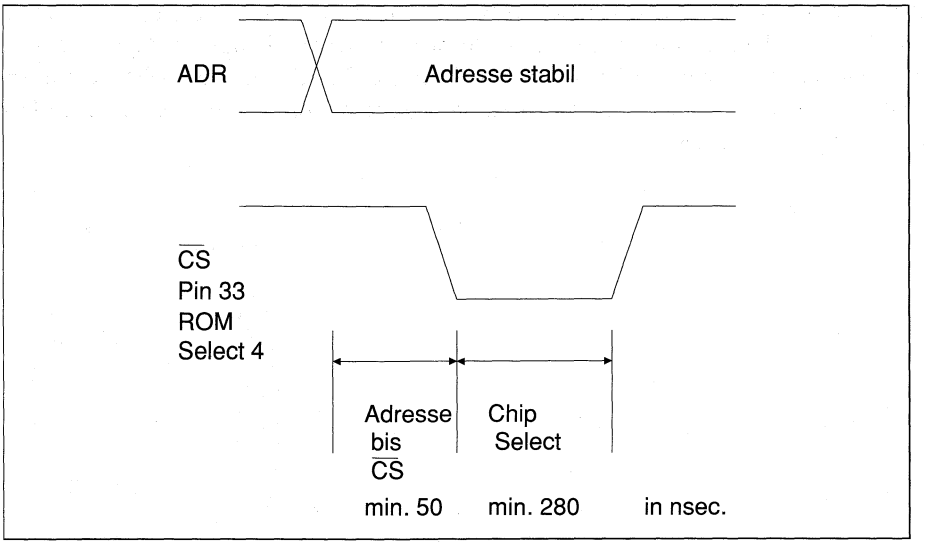


Bild 3.10: Das zeitliche Verhalten ist bei der Entwicklung von Schaltungen von großer Bedeutung

3.6 Parallele sternförmige Verbindung (Beispiel Centronics)

Parallele Verbindungen, die zum Datentransport zwischen einem festen Sender und Empfänger dienen, benötigen keinerlei Adressierungsverfahren. Ein Beispiel eines derartigen Datenverkehrs ist das Leitungssystem nach der Centronics-Norm. Es verbindet in der Regel eine intelligente Einheit mit einer Ausgabestation. Die meisten Drucker werden auf diese Weise an den Computer angeschlossen. Man benutzt hierbei acht Daten- und vier Steuerleitungen zum Datenverkehr in eine Richtung.

Der Sender legt zum Datentransport das Datum auf die Leitungen DATA1 bis DATA8 und kontrolliert, ob der Teilnehmer für die Datenaufnahme bereit ist (BUSY). Sobald die BUSY-Leitung eine logische 0 annimmt, erzeugt der Sender ein STROBE-Signal (kurze logische 1) und schreibt sein Datum in die periphere Einheit. Die angeschlossene Peripherie quittiert durch das ACKNLG-Signal (Acknowledge), daß das Datum eingelesen wurde. Gleichzeitig nimmt der Empfänger ein eventuell aufgetretenes BUSY-Signal ebenfalls zurück und der Sender übergibt das folgende Datum. Die Leitung PE (Paper End) signalisiert: Der Drucker ist am Papierende angelangt. Diese vier Steuerleitungen sind vollständig ausreichend, um den Datentransport abzuwickeln. In einer einfachen Form genügt auch schon das STROBE-

und BUSY-Signal. Ein Bauvorschlag für ein Centronics-Interface befindet sich am Ende dieses Buches. Manche Drucker verfügen über weitere Steuerleitungen, die gewisse Voreinstellungen der Druckfunktion bewirken. Die folgende Tabelle gibt eine Übersicht über die Signal- und Steuerleitungen. Bild 3.11 zeigt die beiden genormten Stecker, die zum Datenverkehr eingesetzt werden.

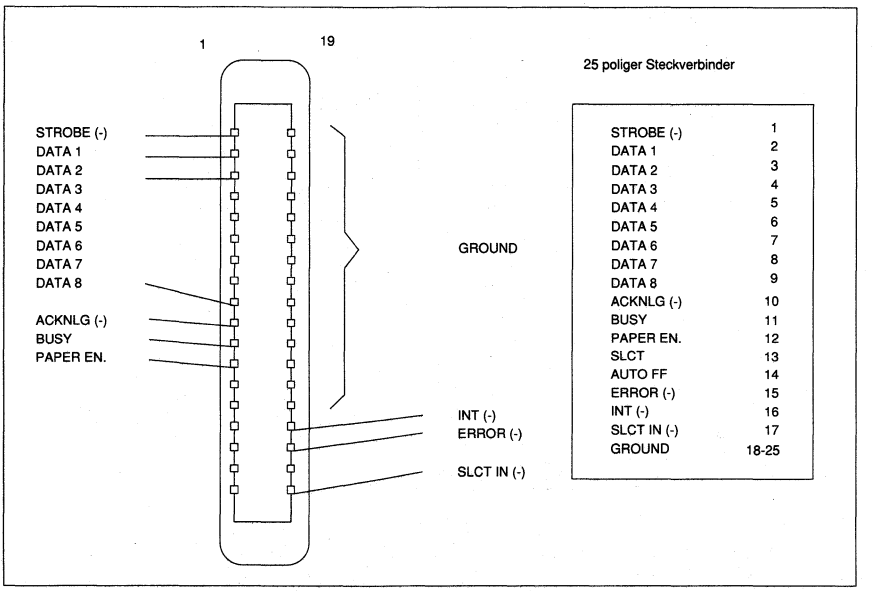


Bild 3.11: Zwei Steckverbinder nach der Centronics-Norm. Links ein 36poliger Stecker, rechts ein 25poliger Sub-D-Stecker.

Signal-Leitung	Signal	Richtung (am Drucker)
1	Strobe (–)	Ein
2	Data 1	Ein
3	Data 2	Ein
4	Data 3	Ein
5	Data 4	Ein
6	Data 5	Ein
7	Data 6	Ein
8	Data 7	Ein
9	Data 8	Ein
10	Acknlg (–)	Aus
11	Busy	Aus

Signal-Leitung	Signal	Richtung (am Drucker)
12	PE	Aus
13	–	
14	Auto Feed (–)	Ein
15	–	
16	0 Volt	
17	Masse Chassis	
18	–	
19 – 30	Masse	
31	Init (–)	Ein (Initialisierung)
32	Fehler (–)	Aus
33	Masse	
34	–	
35	–	
36	Slct In (–)	Ein (Select-Signal)

3.7 Serielle Verbindungen

Mit zunehmenden Entfernungen werden parallele Datenverbindungen immer unrentabler. Durch ein Reduzieren der Leitungen kann hier Abhilfe geschaffen werden. Eine Kommunikation mit nur einer einzigen Leitung (zusätzlich zur Masseleitung) stellt eine häufig realisierte Alternative zu parallelen Verbindungen dar. Der Datentransport erfolgt auf dieser Leitung vom Sender zum Empfänger, wobei die Daten zeitlich sequentiell transportiert werden. Möchte der Empfänger Daten zum Sender schicken, so wird die Transportrichtung nach der ersten Information umgeschaltet und der Transport läuft in umgekehrter Richtung (Half Duplex). Möchten sich zwei Teilnehmer gleichzeitig unterhalten, so brauchen sie zwei Leitungen (Full Duplex), die ihre Datentransportrichtung nicht mehr umschalten müssen (Bild 3.12). Eine Information (Byte oder Wort) wird seriell übertragen, indem sie in ihre Bestandteile (Bits) zerlegt wird und in zeitlicher Folge auf der Leitung erscheint. Deshalb müssen sich beide Teilnehmer vor Beginn der Übertragung auf die Taktrate der Bitsequenz einigen (Baud-Rate). Der Sender legt sich zur Übertragung zuerst ein Byte zurecht und zerhackt dieses in seine Grundbestandteile (Bits). Mit jeder Taktflanke wird dann ein Bit auf den Weg geschickt, wobei der Sender mit dem untersten Bit beginnt (D0) und mit dem obersten (D7) aufhört. Der Empfänger packt die empfangenen Bits wieder zu einem Byte zusammen. Damit sich Sender und Empfänger auf die Dateninformation synchronisieren können, beginnt der Sender immer mit einem Anlaufschritt (Start-Bit) und beendet die Information mit einem (manchmal auch zwei oder 1,5) Sperrschritt (Stop-Bit). Start-Bit und Stop-Bit un-

terscheiden sich immer in ihrem logischen Zustand, damit auch eine Übertragung gleichartiger Zeichen erkannt wird. Bild 3.13 zeigt in einem Timing-Diagramm, wie die Information 35Hex vom Sender zum Empfänger gelangt. Der Empfänger wartet stets auf das Start-Bit und kontrolliert dann normalerweise in der Mitte der Bitsequenz die logische Funktion.

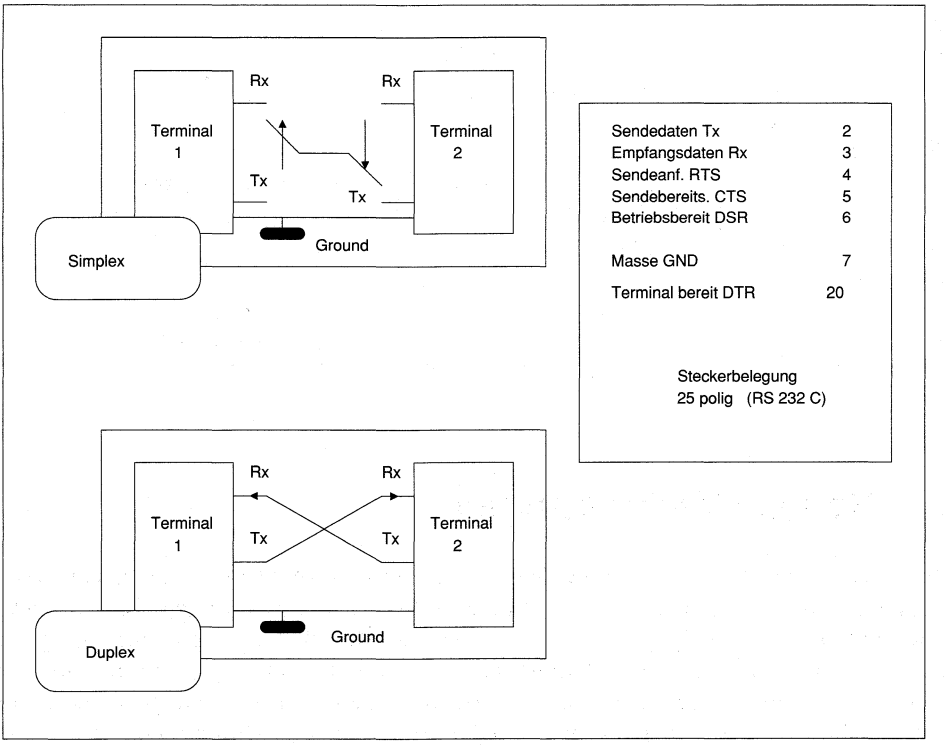


Bild 3.12: Prinzipieller Aufbau der Simplex- und Duplex-Verbindung mit dem dazugehörigen Stecker

Die Baud-Rate gibt an, wie viele Bits pro Sekunde übertragen werden. Bei einer Baud-Rate von 300 erscheint alle 3,33 ms eine neue Information auf der Datenleitung. Mit der ersten Flanke des Start-Bits kontrolliert der Empfänger die erste Dateninformation nach 5 ms (Bit D0 in der Mitte des Datenimpulses). Die zweite Information übernimmt er nach 8,33 ms usw. Bei der seriellen Übertragung zwischen unterschiedlichen Geräten sind normalerweise nur diskrete Übertragungsfrequenzen vorgesehen. So benutzt man Baud-Raten von 110, 300, 600, 1200, 2400, 4800, 9600 und 19200. Höhere Übertragungsraten sind ebenfalls gebräuchlich, wobei dann die gewählten Baud-Raten oftmals von den beteiligten Prozessoren bestimmt werden.

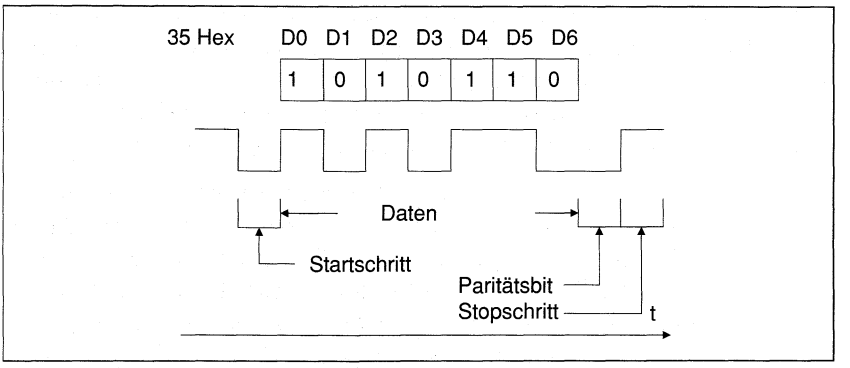


Bild 3.13: Übertragung der Information 35H über eine serielle Leitung

Das in dieser Form vorgestellte Verfahren, wird auch asynchrones Übertragungsverfahren genannt, da der Empfänger eine interne Synchronisation durch die erste Flanke des Start-Bits durchführt. Hinter dem Stop-Bit kann die Synchronisation wieder ausfallen. Im Gegensatz zu diesem Übertragungsverfahren gibt es auch einen Synchronbetrieb, bei dem der Zeitrastergleichlauf über eine ganze Zeichenkette erhalten bleibt. Dazu überträgt der Sender zu Beginn der Datenkette ein Synchronisationsmuster (z.B. einen Wechsel von Nullen und Einsen), auf das der Empfänger mit hoher Genauigkeit einrastet.

3.8 Normungen beim Datentransport

Bei der Datenübertragung bedient man sich festgelegter Normen, welche die gegenseitige Verständigung ermöglichen. Eines der wichtigsten Kodierverfahren ist die ASCII-Norm (American Standard Code for Information Interchange). Diese Norm ordnet jedem Zeichen des (amerikanischen) Alphabets eine spezielle Bitkombination zu, die eine Übertragung im Klartext ermöglicht. Der ASCII-Code wurde von der International Organisation for Standardization (ISO) festgelegt und ist in zahlreiche andere Normen eingeflossen. Insgesamt benutzt man 7 Bit (also 128 Möglichkeiten), um einen Buchstaben, eine Zahl oder ein Steuerzeichen zu kodieren. Bei der Datenübertragung werden in der Regel 8 Bit (plus Start- und Stop-Bit) übertragen, wobei das Bit vor dem Stop-Bit als Paritätsbit zur Datensicherung aufgefaßt wird. Im Anhang befindet sich eine Tabelle, aus der die Kodierung des ASCII-Codes hervorgeht. Die meisten Datenendeinrichtungen (Bildschirme, Drucker usw.) beherrschen neben anderen Kodierverfahren zumindest die ASCII-Norm. Selbst parallele Dateneinrichtungen (z.B. Drucker mit Centronics-Anschluß) bedienen sich dieser Norm. Viele Alphabete beinhalten spezielle Zeichen, die nicht im amerikanischen Zeichensatz vorkommen (im Deutschen z.B. die Umlaute ä, ö, ü, Ä, Ö, Ü und das ß). Um auch diese Zeichen übertragen

zu können, verwendet man unterschiedliche Verfahren. So werden häufig einige selten verwendete ASCII-Zeichen durch die Umlaute ersetzt (z.B. entspricht { dem ä) oder man verwendet statt der Sonderzeichen eine Sequenz von ASCII-Zeichen. Neben dem ASCII-Code sind auch 5-Bit-Codes (z.B. Baudot-Code bei Fernschreibern) oder 8-Bit-Codes (EBCDI-Code; Extended Binary Coded Decimal Interchange Code) weit verbreitet.

Wie die Bitkodierungen, sind auch die Datenpegel bei der Übertragung genau festgelegt. Die gebräuchlichste Schnittstelle zwischen zwei Einheiten stellt die serielle Verbindung nach der CCITT-Empfehlung (Comite Consultativ International Telegraphique et Telephonique) V.24 dar. Bei der V-24-Norm (auch DIN 66020) wird jedem logischen Pegel ein Spannungszustand auf den Datenleitungen zugeordnet, der eine relativ große Datensicherheit gewährleistet. So ergibt eine logische 1 einen Spannungspegel von –3 bis –12 Volt, eine logische 0 entspricht dem Spannungsbereich von +3 bis +12 Volt. Spannungswerte zwischen –3 und +3 Volt sind nicht zulässig. Damit lassen sich Entfernungen bis zu 15 Meter noch sicher überbrücken.

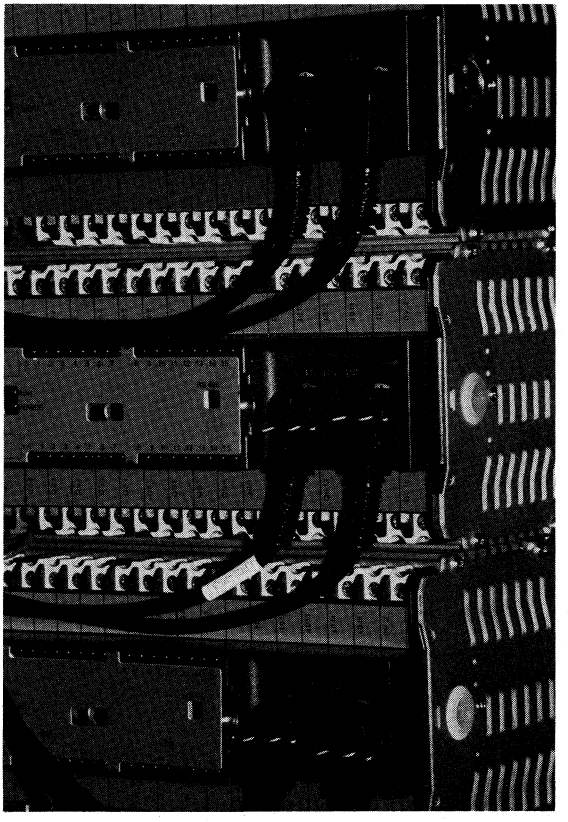


Bild 3.14: »lokales Netz« für die Verbindung von speicherprogrammierbaren Steuerungen

Bei größeren Distanzen bedient man sich häufig der stromgetriebenen Schnittstelle, bei der ein Impulsstrom von ca. 20 mA statt des Spannungspegels eingeprägt wird. Mit wachsender Entfernung und höheren Baud-Raten nimmt der Einsatz der Differentialschnittstellen zu. Zu diesen Schnittstellen gehören die RS 422 und die RS 485 (als Busschnittstelle). Beide Schnittstellen erzeugen auf jeweils zwei Datenleitungen antiparallele Signale, die überaus immun gegen Störungen sind. Die Spannungswerte betragen 0 oder 5 Volt bei Strömen bis 160 mA. Mittels der beiden letztgenannten Schnittstellen lassen sich Entfernungen bis zu einem Kilometer und Datenübertragungsraten bis zu einem Mbaud bewerkstelligen. Neben diesen Übertragungsverfahren kommen in den letzten Jahren zusehends auch Koaxialverbindungen zum Einsatz, die (neben der RS 485) zum Datentransport bei lokalen Netzen (LANs; Local Area Networks) dienen (Bild 3.14).

3.9 Datensicherung

Mit zunehmender Entfernung nimmt die Wahrscheinlichkeit der Datenverfälschung immer mehr zu. Störende Einflüsse wie Reflexionen am Kabelende oder die Einwirkung elektromagnetischer Felder können die Datenübermittlung stören. Der Empfänger ist dabei selten in der Lage, eine Störung festzustellen, da das logische Umklappen einer 0 in eine 1 nicht als Fehler erkannt wird (so wird z.B. eine ASCII-Information »B« durch eine Störung im ersten Datenbit vom Empfänger als ASCII-Information »C« erkannt). Bei der Ansteuerung eines Druckers führen solche Fehler höchstens zu orthographischen Eigenheiten, bei Programmübertragungen können daraus fatale Fehler entstehen. Es ist daher wünschenswert, alle Fehler mit hoher Sicherheit zu erkennen. Im Falle der Datenübertragung kann man dann den Datensatz erneut anfordern und durch den korrekten ersetzen. Im Falle der Datenabspeicherung (Bibliotheken) dagegen, hilft die erneute Anforderung nicht viel, da das Abspeichern eventuell schon Jahre zurückliegt.

Die Fehlererkennungsverfahren teilen sich deshalb in zwei unterschiedliche Strategien auf, die entweder eine Fehlererkennung oder (im zweiten Fall) gar eine Fehlerreparatur zulassen. Im folgenden sind zwei Fehlererkennungsverfahren dargestellt, die bei serieller Datenübertragung angewandt werden und eine bereits recht hohe Datensicherheit garantieren. Selbst noch so große technische Sicherheitsmaßnahmen garantieren keine Fehlerfreiheit; sie helfen lediglich die Fehlerrate zu reduzieren. Deshalb ist es viel sinnvoller, einen Übertragungsfehler mit hoher Sicherheit zu erkennen, als ihn von vorn herein auszuschließen.

Alle Fehlererkennungsverfahren (und Fehlerreparaturverfahren) basieren darauf, der Information eine sogenannte Redundanz hinzuzufügen. Diese Redundanz besteht in der Regel aus überflüssigen Informationen, an denen die Richtigkeit des Informationsgehalts abgeschätzt werden kann. In der einfachsten Form fügt man dem übertragenen Datenbyte jeweils ein Paritätsbit hinzu, das die Quersumme (entweder gerade oder ungerade) der Information wiedergibt. Man verliert durch dieses überflüssige Bit zwar ein wenig an Übertragungszeit,

dennoch lassen sich einfache Fehler (Fehler, bei denen nur ein Bit zerstört wurde) mit Sicherheit erkennen. Das Verfahren scheitert erst, wenn innerhalb eines Zeichens jeweils 2 Bit ihre Information genau umdrehen. Nur in diesem Fall ist die Quersumme wieder identisch zur Ausgangsinformation. Möchte man diesen schon recht seltenen Fehlerfall ebenfalls noch mit Sicherheit erkennen, so bedient man sich in der Regel dem Kreuzsicherungsverfahren, das neben der Paritätserkennung noch über eine Blocksicherung (z.B. Summe über einen kompletten Datensatz) zuläßt. Dieses Verfahren gewährleistet eine sichere Programmübertragung. Der Personalcomputer (auch CP/M-Rechner) verwendet derartige Kreuzsicherungen beim Erstellen und Übertragen sogenannter HEX-Files nach einem Compiler-Lauf. Da bei der Kreuzsicherung jedes Byte einzeln durch die Parität und die Summe aller Bytes durch das Summenbyte abgesichert sind, müssen schon gezielt vier Fehler auftreten, damit die verfälschte Information als richtig erkannt wird.

Im dem vorher dargestellten ISO-Modell entspricht die Fehlererkennung und die Fehlerbehandlung der Schicht 2. Bild 3.15 zeigt einen Informationsblock mit Kreuzsicherung und ein Beispiel einer Störung (von 4 Bit), die nicht mehr als Fehler erkannt werden kann.

Daten	02	4B	34	7A	25	Block-prüfung	
Bit 0	0	1	0	0	1	0	gerade = 0
Bit 1	1	1	0	1	0	1	ungerade = 1
Bit 2	0	0	1	0	1	0	
Bit 3	0	1	0	1	0	0	
Bit 4	0	0	1	1	1	1	
Bit 5	0	0	1	1	0	0	
Bit 6	0	1	0	1	0	0	
Paritätsbit	1	0	1	1	1	0	

Daten	02	4B	34	7A	25	Block-prüfung	
Bit 0	0	1	0	0	1	0	gerade = 0
Bit 1	1	1	0	1	0	1	ungerade = 1
Bit 2	0	0	1	0	1	0	
Bit 3	0	1	0	1	0	0	
Bit 4	0	$\boxed{1}$	1	1	$\boxed{0}$	1	
Bit 5	0	0	1	1	0	0	4facher
Bit 6	0	$\boxed{0}$	0	1	$\boxed{1}$	0	Fehler
Paritätsbit	1	0	1	1	1	0	bleibt unerkannt

Bild 3.15: Bei der Kreuzsicherung bleibt erst ein 4-Bit-Fehler unerkannt

4 Mikroprozessorbausteine

4.1 Allgemeine Betrachtungen

Unter Mikroprozessorbausteinen versteht man nicht nur die Zentraleinheit (CPU), beim Atari ST der 68000, sondern auch andere hochintegrierte Chips, welche die CPU bei den verschiedensten Aufgaben unterstützen und entlasten. Da die Bausteine speziell für Ihre Aufgaben entwickelt wurden, können sie diese wesentlich schneller durchführen als die CPU. Für manche dieser Funktionen wären die meisten Zentraleinheiten, auch eine so leistungsfähige wie der 68000, schlicht überfordert.

So ist es z.B. für die Verbindung eines Computers mit der Außenwelt notwendig, Ein-/Ausgabe-Einheiten (kurz E/A- oder I/O-Einheiten) anzuschließen. Da diese Einheiten im allgemeinen nicht an den Systembus passen, muß hierfür ein Interface verwendet werden. Beispiele für solche Geräte sind Tastatur, Monitor, Drucker, Diskettenlaufwerke oder Festplatten, Modems und alle anderen Geräte, die an einen Computer angeschlossen werden können. Darüber hinaus werden Computer häufig zum Erfassen von Meßdaten oder zum Steuern von Abläufen eingesetzt, wofür ebenfalls eine Interfaceschaltung benötigt wird (wie in diesem Buch beschrieben).

Früher wurden für diese Aufgaben Interface-Karten mit vielen Bauteilen verwendet, die von der Zentraleinheit mit aufwendigen Programmen gesteuert werden mußten. Heute werden für praktisch alle Standardaufgaben spezielle hochintegrierte Bausteine angeboten, die direkt an den Systembus angeschlossen werden. Man nennt diese Bausteine auch Controller oder I/O- Controller. In einem Computer werden neben den oben beschriebenen komplexen Funktionen auch eine Vielzahl einfacher Schaltungen benötigt (wie z.B. in Kapitel 1 behandelt), die in der Vergangenheit ebenfalls durch eine Anzahl diskreter ICs aufgebaut werden mußten. Durch den Fortschritt der Halbleitertechnologie wurde es möglich, immer mehr Funktionen in einem Chip zu integrieren. Ziel dieser Entwicklung ist es, einen Computer aus möglichst wenig Bauteilen aufzubauen. Zum einen sinkt dadurch der Preis für die Hardware erheblich, zum anderen fallen mögliche Fehlerquellen weg. Beim Atari wurden daher fünf sogenannte Custom-Chips entwickelt, wodurch die meisten logischen Bausteine eingespart werden konnten. Diese Bausteine werden im Zusammenhang mit dem Hardwareaufbau des Atari in Kapitel 5 behandelt. Durch alle diese Bausteine wird der Hauptprozessor von vielen Routineaufgaben entlastet, von denen er manche aus Geschwindigkeitsgründen gar nicht

selbst erledigen könnte (z.B. den Datentransfer zwischen Speicher und Disketten- oder Festplattenlaufwerken). Der Mikroprozessor übergibt die Daten z.B. an einen I/O-Baustein, der sie aufbereitet und zum Gerät transportiert. In manchen Fällen wird der Prozessor sogar »stillgelegt« und der externe Baustein übernimmt vollständig die Kontrolle über das System. Dies ist z.B. beim sogenannten DMA (Direct Memory Access, Direkter Speicherzugriff) nötig, der im Zusammenhang mit dem MC 68000 näher erläutert wird.

Man spricht in diesem Zusammenhang auch von »intelligenten« Bausteinen, weil alle Funktionen ohne direkte Kontrolle der Zentraleinheit ausgeführt werden. Viele dieser Bausteine sind auf einfache Weise programmierbar und können für unterschiedliche Anwendungen eingesetzt werden. Es ist daher möglich, mit demselben Chip unterschiedliche Aufgaben zu realisieren. Auf den Hauptplatinen der Atari-Computer sind neben den Custom-Chips auch eine Reihe solcher allgemeiner Mikroprozessorbausteine vorhanden:

- MC 68000 (CPU, Central Processing Unit, Zentraleinheit)
- WD 1772 (FDC, Floppy Disk Controller, Disketten-Baustein)
- MPF 68901 (Multi Function Peripheral, Multifunktions Peripherie-Baustein)
- ACIA 6850 (Asynchronous Communications Interface Adapter, serieller I/O-Baustein)
- PSG 2149 (Programmable Sound Generator, programmierbarer Geräusch-Baustein)

Auf der Interface-Karte befinden sich zwei weitere Mikroprozessorbausteine:

- 8255 (Paralleler I/O-Baustein)
- 8251 (Serieller I/O-Baustein)

Diese Bausteine aus der Intel-Familie wurden aufgrund der niedrigen Kosten und der hohen Verfügbarkeit ausgewählt.

4.2 Die Zentraleinheit MC 68000

Der 68000 wurde ursprünglich von Motorola entwickelt und wird heute von vielen weiteren Firmen als Second Source (Zweithersteller) hergestellt und vertrieben. Er zeichnet sich durch einen sehr einfachen und logischen Aufbau aus, der sowohl für Hard- wie für Software-Entwickler hervorragende Möglichkeiten bietet (wie man am Atari bewundern kann). Dennoch hat sich im professionellen Bereich die Prozessorlinie des 8088, 80286 usw. von Intel stärker durchgesetzt. Doch dies hat kaum etwas mit der Leistungsfähigkeit dieser Prozessoren zu tun, sondern eher mit geschäftlichen Interessen. Der Durchbruch der Intel-Familie ist auf die Entscheidung von IBM zurückzuführen, den damals schon veralteten 8088 in den PC (Personal Computer) einzubauen und mit der anschließenden weltweiten Verbreitung dieses Computertyps. Zum Bedauern von IBM und zur Schadenfreude aller anderen trugen zu dieser Verbreitung vor allem auch die Nachbauer bei. Zur Entscheidung IBMs für den 8088 sei hier noch die Zusammenarbeit mit Intel angemerkt.

In diesem Kapitel soll der Hardwareaufbau und die allgemeine Funktionsweise beschrieben werden, soweit er zum Verständnis der Funktionsweise des Atari ST notwendig ist. Die Programmierung des Prozessors wird schließlich in Kapitel 7.4 beschrieben. Beide Kapitel sollen nur der ersten Orientierung dienen und können ein spezielles Buch über den 68000 nicht ersetzen.

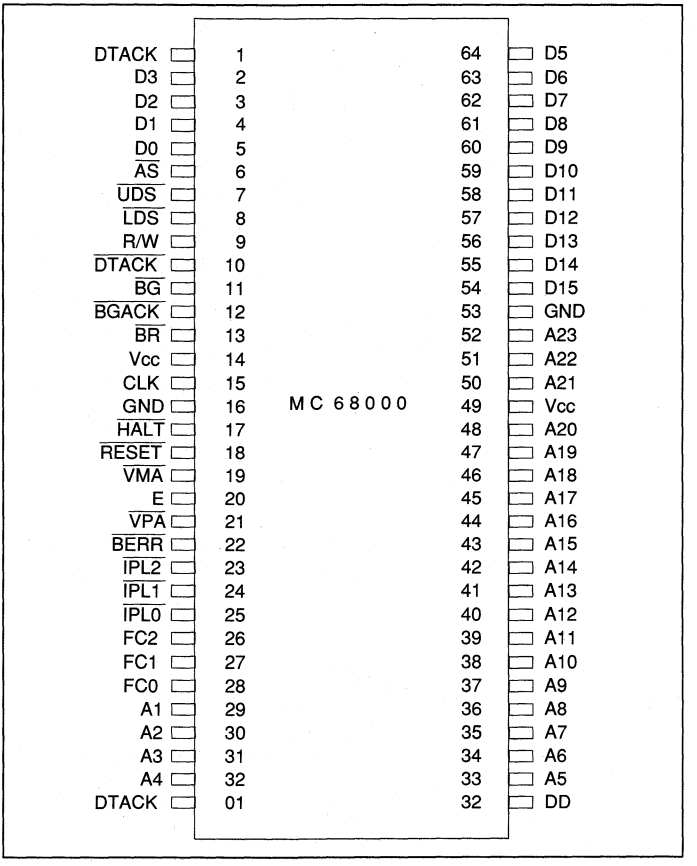


Bild 4.1: Die Anschlußbelegung des 68000

Bild 4.1 zeigt die Anschlußbelegung des 68000, der eine echte 16-Bit-CPU ist, da er 16 Datenleitungen hat. Alle internen Register sind sogar 32 Bit breit, weswegen manche Leute auch von einer 32-Bit-CPU sprechen. Der Adreßbus umfaßt 23 Leitungen von A1 bis A23, damit lassen sich 16 Mbyte direkt adressieren, also ohne Umwege über irgendwelche Segmentregister (wie bei der 8088-Familie). Da der Datenbus 16 Bit breit ist. wird die Adreßleitung A0 nicht benötigt. Damit kann aber nicht auf ungerade Adressen zugegriffen

werden, denn der Speicher ist ja nach wie vor byteweise organisiert. Um Operationen auf ungerade Adressen zu ermöglichen hat der 68000 zwei Steuerleitungen UDS (Upper Data Strobe, Oberer Daten Signalpegel) und LDS (Lower Data Strobe, Unter Daten Signalpegel), beide aktiv Low.

Tabelle der Anschlüsse beim 68000

Anschluß	**E/A**	**Funktion**
A1 – A23	A	Adreßleitungen
D0 – D15	E/A	Datenleitungen
AS*	A	Adress Strobe, Adreßfreigabe
R/W*	A	Read/Write, Lesen/Schreiben
UDS*	A	Upper Data Strobe, obere Datenfreigabe
LDS*	A	Lower Data Strobe, untere Datenfreigabe
DTACK*	E	Data Acknowledge, Datentransferbestätigung
BR*	E	Bus Request, Busanforderung
BG*	A	Bus Grant, Buszuteilung
BGACK*	E	Bus Grant Acknowledge, Buszuteilungsbestätigung
IPL0* – IPL2*	E	Interrupt Priority Level, Interruptprioritätsebene
BERR*	E	Bus Error, Busfehler
RESET*	E/A	Reset, Rücksetzen
HALT*	E/A	Halt, Anhalten
E	A	Synchrontakt, 1/10 des Systemtakts
VMA*	A	Valid Memory Adress, gültige Seicheradresse
VPA*	E	Valid Peripheral Adress, gültige Peripherieadresse
FCO – FC2	A	Function Code, Funktionscode
CLK	E	Clock, Systemtakt
Vcc	–	Spannungsleitung, +5 V
GND	–	Ground, Erde

Alle mit einem "*" versehenen Signale sind Aktiv Low. Die Spalte E/A zeigt an, ob die Leitungen als Eingang (E), Ausgang (A) oder beides dienen.

Der 68000 kann in zwei verschiedenen Betriebsarten arbeiten, im User und Supervisor Mode. Wie der Name schon andeutet, ist der Supervisor Mode für das Betriebssystem vorgesehen. Nur in dieser Betriebsart stehen alle Befehle des 68000 zur Verfügung. Die Anwenderprogramme laufen üblicherweise im User Mode, in dem alle sogenannten privilegierten Befehle nicht erlaubt sind. Durch einen geeigneten Hardwareaufbau können Programme und Daten für User und Supervisor in getrennte Bereiche des Speichers gelegt werden.

Der 68000 hat die drei Function-Code-Leitungen (Funktionsleitungen) FC0 bis FC2, auf denen er externen Baugruppen mitteilt, in welchem Status bzw. Zustand er sich gerade befindet.

FC2	FC1	FC0	Bedeutung
0	0	0	nicht definiert (reserviert)
0	0	1	USER-Daten
0	1	0	USER-Programm
0	1	1	nicht definiert
1	0	0	nicht definiert
1	0	1	Supervisor-Daten
1	1	0	Supervisor-Programm
1	1	1	Interrupt-Verarbeitung

Mit den Function-Code-Leitungen und einem geeigneten Hardware-Aufbau kann eine angeschlossene Logik erkennen, ob der Prozessor gerade im User oder Supervisor Mode arbeitet und ob in dieser Betriebsart der Zugriff auf diesen Speicherbereich erlaubt ist. Bei einer unerlaubten Speicheroperation kann z.B. ein Busfehler (Signal BERR) und ein entsprechender Interrupt ausgelöst werden.

An den CLK-Eingang wird eine Frequenz angelegt, die alle internen Operationen der CPU steuert. Der 68000 kann mit 4 bis 16 MHz betrieben werden, im Atari sind es 8 MHz. Damit ist jeder Maschinenzyklus 125 ns lang.

Das I/O-Konzept des 68000

Beim 68000 gibt es keine speziellen Leitungen und Befehle, die über die Peripherieeinheiten angesprochen werden können (wie etwa beim 8088), sondern dies geschieht genauso, als würde eine Speicheroperation durchgeführt. Hierfür wird in einem Computersystem normalerweise ein Teil des Speichers als I/O- Bereich reserviert (beim Atari die Adressen von $FF8000 bis $FFFC07). Wird eine Adresse aus diesem Bereich auf den Bus gelegt, so dekodiert eine spezielle Hardware dies und selektiert die entsprechende Peripherie-Einheit. Die Daten von und zu dieser Einheit laufen ganz normal über den Datenbus. Man nennt dieses Verfahren auch Memory Mapped I/O (im Speicherraum befindlicher E/A). Ein Beispiel soll dies verdeutlichen:

Soll z.B. ein Datenbyte an den Drucker geschickt werden, der beim Atari ST auf der Adresse $FF8802 liegt, so legt die CPU die Adresse $FF8802 auf den Adreßbus und das Datenbyte auf den Datenbus. An der Adresse $FF8802 ist nun aber kein Speicher-Chip angeschlossen, sondern die Adressen von $FF8000 bis $FFFC07 (nur beim Atari ST, bei anderen 68000-Computern kann der I/O-Bereich an einer anderen Stelle liegen) werden von einer Dekodierlogik überwacht, die immer dann anspricht, wenn eine Adresse aus diesem Bereich auf dem Bus liegt. Diese Logik aktiviert durch ein Signal den Sound-Chip, der auch über einen Parallelport verfügt, der beim Drucken die Daten an den Drucker schickt.

Die Dekodierlogik für den I/O-Bereich liegt beim Atari ST im sogenannten Glue, einem der Custom-Chips (siehe Kapitel 5).

Das DMA-Konzept des 68000

Bei einer DMA-Operation (Direct Memory Access, Direkter Speicherzugriff) wird der Speicher nicht von der CPU, sondern von einem anderen Mikroprozessorbaustein gelesen oder beschrieben.

Man hat hierdurch die Möglichkeit, Daten ohne Kontrolle der CPU zwischen einer schnellen Peripherie-Einheit (z.B. Diskettenlaufwerk oder Festplatte) und dem Hauptspeicher zu transferieren. Während dieser Zeit übernimmt ein sogenannter DMA-Baustein die Kontrolle über den Bus. Hierfür muß die CPU für eine kurze Zeit quasi ausgeschaltet werden (in einen hochohmigen Zustand übergehen, den man auch Tristate-Zustand nennt).

Während dieser Zeit kann der DMA-Controller über den Bus verfügen. Die CPU muß sich vom Bus trennen, damit nicht gleichzeitig zwei Bus Master vorhanden sind, die Signale auf den Bus legen, was zu falschen Spannungen auf den Leitungen führen würde. Um dies zu erreichen, hat der 68000 mehrere Steuerleitungen.

Mit dem Signal BR (Bus Request, Busanfrage, aktiv Low) leitet der DMA-Prozessor einen DMA-Zyklus ein. Sobald dieses Signal auf Low geht, beendet die CPU den gerade bearbeiteten Befehl, und deaktiviert die Signale AS, DTACK, und eventuell BGACK. Erst dann wird das Signal BGACK (Bus Grant Acknowledge, Buszuteilung akzeptiert, aktiv Low) vom anfordernden Baustein aktiviert, der damit den Bus übernommen hat. Dieses Signal bleibt solange aktiv, bis der DMA-Zyklus beendet ist. Während dieser Zeit verharrt der 68000 in einem quasierstarrten Zustand. Alle aktiven Leitungen sind hochohmig, d.h. vom Bus getrennt.

Sobald das BGACK-Signal vom DMA-Prozessor zurückgenommen wird, erwacht der 68000 zu neuem Leben und übernimmt die Kontrolle über das System.

Das Interrupt-Konzept des 68000

Die Interrupts beim 68000 heißen eigentlich Exceptions (Ausnahme, nach der Motorola-Nomenklatur), aber da der Begriff Interrupt auch bei anderen Prozessoren weit verbreitet ist, soll er hier beibehalten werden. Der Aufbau der Interruptstruktur ist in seiner Grundanlage ähnlich wie bei anderen Prozessoren (z.B. 8086).

Ein Interrupt kann entweder durch eine Leitung an der CPU oder durch einen Maschinenbefehl ausgelöst werden. In diesem Kapitel sollen nur die Hardware-Interrupts behandelt werden. Die Software-Interrupts werden ausführlich in Kapitel 7 beschrieben. Sie funktionieren vom Prinzip her aber genauso wie die Hardware-Interrupts.

Der 68000 erlaubt 255 verschiedene Interrupts, denen am Beginn des Speichers eine Tabelle mit Interrupt-Vektoren zugeordnet ist. Jeder Interrupt hat eine Nummer (1 – 255), zu der ein Eintrag in der Tabelle gehört. Jeder Eintrag (bis auf den ersten) ist vier Byte lang und enthält die Adresse (daher der Begriff Vektor), bei der sich das Programm zum Verarbeiten des

Interrupts befindet. Die Tabelle liegt im Speicher also von der Adresse $0000 bis $03ff (der letzte Vektor beginnt bei der Adresse $03fc).

Wird z.B. der Interrupt Nummer 20 ausgelöst, so holt der 68000 den Interrupt-Vektor Nr. 20, der ab der Adresse $54 steht und lädt den Befehlszähler mit den Werten der Adressen $54 – $57 (Berechnung: Interrupt-Nr * 4 + 4). Mit dem nächsten Befehl wird dann das Unterprogramm zur Bearbeitung des Interrupts ausgeführt. Das Programm steht also nicht an der Adresse $54, sondern dort steht nur die Adresse, an der sich dieses Programm befindet.

Die Adresse 0 hat eine besondere Bedeutung. Hier ist der RESET-Vektor untergebracht, der aus 8 Byte besteht, während alle anderen Vektoren nur aus 4 Byte bestehen. Ab der Adresse $0000 steht der Startwert des Supervisor-Stacks, ab der Adresse $0004 die Anfangsadresse des Betriebssystems. Praktisch ist dies die Adresse, wo der 68000 nach dem Einschalten bzw. nach einem Reset mit der Programmausführung beginnt.

Somit ist auch verständlich, wieso bei der Berechnung der Interrupt-Vektoren aus den Interrupt-Nummern eine 4 addiert werden muß.

Zum Auslösen eines Interrupts gibt es beim 68000 die drei Leitungen IPL0 bis IPL2 (Interrupt Priority Level, Priorität der Interruptebene, aktiv Low), welche gleichzeitig auch die Priorität des Interrupts bestimmen. Sind alle drei Anschlüsse inaktiv (High), so wird gerade kein Interrupt bearbeitet, ist dagegen mindestens einer der Anschlüsse aktiv (Low), so liegt ein Interrupt an. Die Nummern 1 bis 6 sind maskierbare Interrupts, während die Nummer 7 ein nichtmaskierbarer Interrupt (NMI, Non Maskable Interrupt) ist, der also bei Auslösung immer ausgeführt werden muß.

Durch die beiden Leitungen VPA und DTACK werden die Interrupts des 68000 in zwei Kategorien unterteilt:

VPA	(Valid Peripheral Adress, aktiv Low)	–	Autovektoren
DTACK	(Data Acknowledge, aktiv Low)	–	Non-Autovektoren

Bei den sogenannten Autovektoren werden die Nummern, die an den Leitungen IPL0 bis IPL2 anliegen, gleich in eine Interruptnummer umgerechnet. Diesen Autovektoren sind in der Interrupt-Tabelle die Nummern 25 bis 31 zugeordnet (25 = $0064 = Nr. 1 bis 31 = $007c = Nr. 7).

Bei den Non-Autovektoren muß die Vektornummer vom auslösenden Baustein über die unteren 8 Bit des Datenbus an die CPU übergeben werden, die daraus über die Tabelle die Interruptadresse berechnet und das Interrupt-Programm anspringt. Den Non-Autovektoren sind die Nummern 64 bis 255 (Adressen $100 bis $3FC) zugeordnet.

4.3 Der Multifunktionsbaustein MFP 68901

In diesem Kapitel soll ein weiteres Mitglied der 68000er-Familie besprochen werden, der im Atari seinen Dienst versieht: der MFP 68901 (Multi Function Peripheral, Multifunktions-Peripherie-Baustein).

Wie der 68000 selbst zeichnet sich auch dieser Baustein durch eine große Leistungsvielfalt aus. Es sind folgende Funktionen integriert:

- ein 8-Bit-Parallel-Port
- eine serielle Schnittstelle
- vier universelle Zähler

Die Leitungen der parallelen Schnittstelle können einzeln gesetzt werden, die Zähler sind in unterschiedlichen Betriebsweisen zu verwenden. Die serielle Schnittstelle kann über DMA betrieben werden, was beim Atari aber nicht geschieht.

Der MFP ist in der Lage, 16 verschiedene Interrupts auszulösen, die genau auf die Interrupt-Struktur des 68000 zugeschnitten sind.

Tritt an einem der 16 interruptfähigen Kanälen des MFP ein Ereignis auf, so wird beim 68000 ein Non-Autovektor-Interrupt ausgelöst. Das bedeutet also, der Baustein legt die Nummer des Interrupts für den 68000 auf die unteren 8 Bit des Datenbusses (wie die Nummer erzeugt wird, ist beim Vektor Register des MFP beschrieben). Der MFP kann bei der Interruptverarbeitung in zwei Modi betrieben werden:

- SEI, Software End of Interrupt, Software-Interrupt-Ende
- AEI, Automatic End of Interrupt, automatisches Interrupt-Ende

Beim SEI-Modus muß nach dem Auslösen eines Interrupts dieser Interruptkanal durch einen Befehl (Software) wieder freigegeben werden. Beim AEI-Modus wird der Kanal wieder freigegeben, sobald der Baustein die Interrupt-Nummer an die CPU übergeben hat.

Alle Timer bestehen aus einem Datenregister, einem programmierbaren Vorteiler und einem 8-Bit-Abwärtszähler. Am Baustein hat jeder Timer eine Ausgangsleitung (TAO bis TDO). Die Timer A und B verfügen darüber hinaus auch noch über jeweils eine Eingangsleitung (TAI und TBI).

Die Timer A und B können in drei verschiedenen Betriebsarten arbeiten:

- Delay Mode, Verzögerungs-Modus
- Event Count Mode, Ereignis-Zähl-Modus
- Pulse Width Mode, Pulsweiten-Modus

Im Delay Mode wird der Inhalt des Zählers bei jedem Impuls des Vorteilers um 1 verringert. Sobald er den Wert 0 erreicht, wechselt der Ausgang des Timers seinen Pegel. Gleichzeitig

wird der Abwärtszähler mit dem Inhalt des entsprechenden Datenregisters neu geladen. In diesem Modus erhält man an den Ausgängen der Timer also Rechteckimpulse.

Im Event Count Modus können an den Timer-Eingängen (TAI und TBI) anliegende Impulse gezählt werden. Die Grenzfrequenz (das ist die maximale Zählfrequenz) darf ein Viertel der Betriebsfrequenz des 68901 nicht überschreiten.

Im Pulse Width Mode können die Längen von Pulsen gemessen werden. Hierfür werden die beiden Eingänge zum Ein- und Ausschalten der beiden Zähler verwendet. Die Timer C und D können nur im Delay Mode arbeiten.

Zur Programmierung besitzt der MFP 24 Register, deren Bedeutung im folgenden beschrieben wird. Gleichzeitig wird auch die Adresse angegeben, unter der das entsprechende Register beim Atari zu erreichen ist (siehe Speicheraufbau in Kapitel 5). In der folgenden Liste ist neben der Speicheradresse des Atari ST, an der sich die Register des 68901 befinden, eine kurze Beschreibung angegeben.

$FFFA01 GPIP, General Purpose I/O Register, Datenregister des Parallelports
Je nachdem wie die Leitungen des Parallel-Ports geschaltet sind, werden die Daten an diesem Port gelesen oder geschrieben.

$FFFA03 AER, Active Edge Register, Register für die aktive Flanke
Wenn die Leitung des Parallel-Ports auf Eingabe programmiert sind, kann jede Leitung einen Interrupt auslösen. Mit diesem Register wird festgelegt, ob der Interrupt bei steigender oder fallender Flanke ausgelöst wird. Jedem Bit ist eine Leitung zugeordnet.

0 Interrupt bei steigender Flanke
1 Interrupt bei fallender Flanke

$FFFA05 DDR, Data Direction Register, Datenrichtungsregister
Mit diesem Port wird die Richtung jeder Leitung am Parallel-Port festgelegt. Jedem Bit ist eine Leitung zugeordnet.

Bit 0 —> Leitung 0

0 Eingang 1 Ausgang

Bit 7 —> Leitung 7

$FFFA07 IERA, Interrupt Enable Register A, Inerrupt-Steuer-Register
Der Baustein kann Interrupt aus 16 verschiedenen Kanälen auslösen. Jeder dieser möglichen Interrupt-Kanäle kann mit den Interrupt Enable Registern A und B einzeln ein- und ausgeschaltet werden.

0 Interrupt verboten
1 Interrupt erlaubt

Den einzelnen Bits dieses Registers sind folgende Interrupt-Kanäle zugeordnet:

Bit 0: Timer B
Bit 1: Transmitter-Fehler
Bit 2: Transmitter-Puffer leer
Bit 3: Empfänger-Fehler
Bit 4: Empfänger-Puffer voll
Bit 5: Timer A
Bit 6: Parallel-Port Bit 6
Bit 7: Parallel-Port Bit 7

$FFFA09 IERB, Interrupt Enable Register B
Beschreibung wie Interrupt Enable Register A. Für dieses Register gelten folgende Zuordnungen:

Bit 0: Parallel-Port Bit 0
Bit 1: Parallel-Port Bit 1
Bit 2: Parallel-Port Bit 2
Bit 3: Parallel-Port Bit 3
Bit 4: Timer D
Bit 5: Timer C
Bit 6: Parallel-Port Bit 4
Bit 7: Parallel-Port Bit 5

$FFFA0B IPRA, Interrupt Pending Register A, Interrupt-Anzeige-Register A
Die Interrupt Pending Register A und B zeigen an, an welchem Kanal ein Interrupt aufgetreten ist. Das Löschen dieser Bits hängt vom Interrupt-Modus ab, in dem der MFP betrieben wird. Im SEI-Modus muß das Bit durch eine Schreiboperation auf dieses Register gelöscht werden, im AEI-Modus dagegen wird es automatisch gelöscht, sobald die Interrupt-Nummer übergeben wurde. Die Zuordnung der Bits zu den Kanälen ist genau so wie bei den beiden Interrupt Enable Registern.

0 kein Interrupt
1 ein Interrupt ist aufgetreten

$FFFA0D IPRB, Interrupt Pending Register B
Beschreibung wie beim Interrupt Pending Register A.

$FFFA0F ISRA, Interrupt In Service Register A
Die Funktion dieses Registers hängt ebenfalls davon ab, in welchem Interrupt-Modus der MFP betrieben wird. Im SEI-Modus wird das entsprechende Bit gesetzt, wenn die Vektor-Nummer übergeben wird und muß am Ende der Interrupt-Verarbeitung durch die Software gelöscht werden. Beim ASI-Modus wird das entsprechende Bit automatisch gelöscht.

Solange ein Bit in den ISR-Registern gesetzt ist, sind alle Interrupts mit niedriger Priorität ausgeblendet. Die Zuordnung der Bit zu den Kanälen ist genau so wie bei den beiden Interrupt Enable Registern.

0 kein Interrupt
1 Interrupt wird bearbeitet

$FFFA11 ISRB, Interrupt in Service Register B
Beschreibung wie beim Interrupt in Service Register A.

$FFFA13 IMRA, Interrupt Mask Register A
Mit diesem Register können eingeschaltete Interrupt-Kanäle ausgeblendet (man sagt auch ausmaskiert) werden. Die Zuordnung der Bits zu den Kanälen ist genau so wie bei den beiden Interrupt Enable Registern.

0 Interrupt-Kanal wegmaskiert
1 Interrupt-Kanal freigegeben

$FFFA15 IMRB, Interrupt Mask Register B
Beschreibung wie beim Interrupt Mask Register A.

$FFFA17 VR, Vektor Register
Wenn beim Auftreten eines Interrupts eine Vektor-Nummer übergeben wird, so ist in den unteren vier Bits dieser Zahl der Interrupt-Kanal kodiert. Die oberen 4 Bit werden aus den oberen 4 Bit des Vektor-Register kopiert. Damit können die Interrupts des MFP an verschiedene Stellen der Interrupt-Tabelle des 68000 geschoben werden. Bit 0 bis 2 werden nicht benutzt und Bit 3 legt den Interrupt-Modus des MFP fest:

0 AEI, Automatic End of Interrupt
1 SEI, Software End of Interrupt

$FFFA19 TACR, Timer A Control Register
Mit diesem Register wird die Funktionsweise von Timer A programmiert, wobei die einzelnen Bits folgende Bedeutung haben:

Bit: 3 2 1 0	**Funktion**
0 0 0 0	Timer anhalten, keine Funktion ausführen
0 0 0 1	Delay Modus, Vorteiler durch 4
0 0 1 0	Delay Modus, Vorteiler durch 10
0 0 1 1	Delay Modus, Vorteiler durch 16
0 1 0 0	Delay Modus, Vorteiler durch 50

Bit: 3 2 1 0	Funktion
0 1 0 1	Delay Modus, Vorteiler durch 64
0 1 1 0	Delay Modus, Vorteiler durch 100
0 1 1 1	Delay Modus, Vorteiler durch 200
1 0 0 0	Event Count Modus
1 0 0 1	Pulsweiten Modus, Vorteiler durch 4
1 0 1 0	Pulsweiten Modus, Vorteiler durch 10
1 0 1 1	Pulsweiten Modus, Vorteiler durch 16
1 1 0 0	Pulsweiten Modus, Vorteiler durch 50
1 1 0 1	Pulsweiten Modus, Vorteiler durch 64
1 1 1 0	Pulsweiten Modus, Vorteiler durch 100
1 1 1 1	Pulsweiten Modus, Vorteiler durch 200

Bit 4: Hiermit kann der Ausgang des Timers von außen auf Null gelegt werden.

$FFFA1B TBCR, Timer B Control Register
Beschreibung wie beim Timer A Control Register.

$FFFA1D TCDCR, Timer C and D Control Register
Mit diesem Register werden die Funktionen von Timer C und D gesteuert. Hierbei sind die Bits 0 bis 2 für den Timer D und die Bits 4 bis 6 für den Timer C vorgesehen. Die Bitkombinationen sind für beide Timer gleich.

Bit: 2 1 0	Funktionen für Timer D
Bit : 6 5 4	**Funktionen für Timer C**
0 0 0	Timer anhalten, keine Funktion ausführen
0 0 1	Delay Modus, Vorteiler durch 4
0 1 0	Delay Modus, Vorteiler durch 10
0 1 1	Delay Modus, Vorteiler durch 16
1 0 0	Delay Modus, Vorteiler durch 50
1 0 1	Delay Modus, Vorteiler durch 64
1 1 0	Delay Modus, Vorteiler durch 100
1 1 1	Delay Modus, Vorteiler durch 200

$FFFA1F TADR, Timer A Data Register
Dieses Register wird mit dem Wort geladen, von dem heruntergezählt wird und das nach einem Impuls am Ausgang wieder in den Zähler kopiert wird.

$FFFA21 TBDR, Timer B Data Register
Beschreibung wie beim Timer A Data Register.

$FFFA23 TCDR, Timer C Data Register
Beschreibung wie beim Timer A Data Register.

$FFFA25 TDDR, Timer D Data Register
Beschreibung wie beim Timer A Data Register.

$FFFA27 SCR, Synchronous Character Register
In dieses Register muß vor dem Beginn eines synchronen Datentransfers das Zeichen geschrieben werden, das den Beginn der Übertragung kennzeichnet.

$FFFA29 UCR, USART Control Register
Mit diesem Register werden die Betriebsparameter der seriellen Schnittstelle festgelegt.

Bit 0:	nicht benutzt.
Bit 1:	Parität: 0 = ungerade, 1 = gerade.
Bit 2:	Parität: 0 keine Parität, 1 wie in Bit 1.
Bit 3 u. 4:	Datenformat.

Bit: 4 3	**Bedeutung der Datenformate**
0 0	0 Startbit, 0 Stopbit, synchron
0 1	1 Startbit, 1 Stopbit, asynchron
1 0	1 Startbit, 1.5 Stopbit, asynchron
1 1	1 Startbit, 2 Stopbit, asynchron

Bit 5 u. 6: Bits pro Zeichen

Bit: 6 5	**Länge eines übertragenen Worts**
0 0	8 Bit
0 1	7 Bit
1 0	6 Bit
1 1	5 Bit

Bit 7: Clock Mode, 0 = Frequenz vom TC und RC, bestimmt die Übertragungsrate
1 = Frequenz geteilt durch 16 bestimmt die Übertragungsrate.

$FFFA2B RSR, Receiver Status Register
Dieses Register zeigt den Zustand des Empfängers der seriellen Schnittstelle an. Die einzelnen Bits haben folgende Bedeutung:

Bit 0: RE, Receiver Enable, Empfänger einschalten
0 = Empfänger ausgeschaltet.

Bit 1: SS, Synchron Strip Enable, bei gelöschtem Bit gelangen Synchronzeichen nicht in den Empfangspuffer.

Bit 2: M/CIP, Match/Character in Progress,
synchron: Synchronzeichen empfangen, asynchron: Startbit empfangen.

Bit 3: F/SorB, Found, Search or Break detected,
synchron: Synchronzeichen empfangen, asynchron: Break empfangen.

Bit 4: FE, Frame Error, Anzahl Stopbits stimmt nicht.

Bit 5: PE, Parity Error, Paritätsfehler.

Bit 6: OE, Overrun Error, Zeichen überlesen.

Bit 7: BF, Buffer Full, Zeichen im Puffer.

$FFFA2D TSR, Transmitter Status Register
Dieses Register zeigt den Zustand des Senders der seriellen Schnittstelle an. Die einzelnen Bits haben folgende Bedeutung:

Bit 0: TE, Transmitter Enable, Sender einschalten,
0 = Sender ausgeschaltet.

Bit 1 u. 2: Zustand des Ausgangs bei ausgeschaltetem Sender.

Bit 3: Break, synchron: keine Funktion.
asynchron: 1 = Break senden.

Bit 4: End of Transmission, Sender gesperrt

Bit 5: AT, Auto Turnaround, 1 = Empfänger wird automatisch eingeschaltet, wenn der Sender ausgeschaltet wird.

Bit 6: UE, Underrun Error, Unterlauffehler.

Bit 7: BE, Buffer Empty, Puffer leer.

$FFFA2F UDR, USART Data Register
Über dieses Register werden die Daten übertragen. Ein hineingeschriebenes Zeichen wird über die serielle Schnittstelle gesendet, ein empfangenes Zeichen kann hier ausgelesen werden.

4.4 Der Disketten-Controller WD 1772

Der 1772 von Western Digital dient zur Steuerung des Diskettenlaufwerks. Hierfür verfügt der Baustein über Anschlüsse, die z.B. den Motor des Laufwerks ein- und ausschalten, anzeigen ob der Track 00 erreicht wurde oder den Schreibschutz erkennen. Zur Programmierung hat der 1772 fünf Register, die durch zwei Adreßleitungen und eine R/W-Leitung angesprochen werden können.

A0	A1	R/W	Port-Name
0	0	0	Status
0	0	1	Kommando
0	1	x	Track
1	0	x	Sektor
1	1	x	Daten Anmerkung: x steht für einen beliebigen Zustand (1 oder 0)

Command Register, Kommandoregister
In dieses Register werden die Befehle an der FDC geschrieben, von denen es elf gibt:

Nr. Typ	Befehl mit Erläuterung
1.	Restore, Spur Null anfahren
2.	Seek, Spur suchen
3.	Step, eine Spur in die bisherige Richtung fahren
4.	Step in, eine Spur nach innen fahren
5.	Step out, eine Spur nach außen fahren
6.	Read Sector, einen Sektor lesen
7.	Write Sector, einen Sektor schreiben
8.	Read Adress, Adressen-Identifikation lesen
9.	Read Track, eine Spur lesen
10.	Write Track, eine Spur schreiben
11.	Force Interrupt, Interrupt auslösen

Alle diese Befehle sind in den oberen drei oder vier Bit des Kommandowortes kodiert. Mit den übrigen Bits werden Parameter übergeben. Es würde diese Darstellung sprengen, alle diese Befehle im Detail zu behandeln.

Status Register, Statusregister
Die einzelnen Bits dieses Registers zeigen den Zustand des FDC an und haben folgende Bedeutung:

Bit 0: Busy, beschäftigt. Es wird gerade ein Kommando ausgeführt.

Bit 1: Data Request/Index, Datenanforderung/Index. Beim Lesen: Datenregister ist voll, muß also ausgelesen werden; beim Schreiben: Datenregister ist leer, muß also neu gefüllt werden.

Bei Kommandos vom Typ 1 wird durch dieses Bit der Index-Impuls angezeigt.

Bit 2: Lost Data/Track 0, Daten verloren.

Bit 3: CRC Error, Prüfsummen-Fehler. Die Ursache des Fehlers wird in Bit 4 angezeigt.

Bit 4: Record not found, Datensatz nicht gefunden.

Bit: 4 3	**Bedeutung**
0 0	kein Fehler
1 1	Fehler im ID-Bereich der Diskette
0 1	Fehler im Daten-Bereich der Diskette

Bit 5: Record Type/Spin Up, Datensatz-Typ, Drehzahl erreicht. Bei Positionierbefehlen: Der Motor hat seine Nenndrehzahl erreicht. Sonst: Bei gesetztem Bit wurde eine Daten-Adreß-Markierung gefunden.

Bit 6: Write Protect, schreibgeschützt. Bei Schreibzugriffen zeigt dieses Bit eine schreibgeschützte Diskette an.

Bit 7: Motor on, Motor ein.

Track Register, Spurregister
Im Spurregister ist die aktuelle Spur des Schreib-/Lesekopfs gespeichert. Bei jedem Spurwechsel wird dieser Wert aktualisiert. Das Spurregister kann sowohl gelesen als auch beschrieben werden.

Sektor Register, Sektorregister
Das Sektorregister enthält bei allen Schreib- und Leseoperationen die Sektornummer auf die der Befehl angewendet werden soll.

Date Register, Datenregister
Über das Datenregister werden alle Daten zwischen Diskettenlaufwerk und Hauptspeicher ausgetauscht. D.h., die von der Diskette gelesenen Daten müssen von diesem Register abgeholt werden und die auf die Diskette zu schreibenden Werte werden an dieses Register übergeben.

Beim Atari wird der 1772 über den DMA-Chip angesprochen und belegt daher im Adreßbereich nur eine Adresse ($FF8604). Welches Register angesprochen wird, muß vorher festgelegt werden.

4.5 Der serielle Baustein ACIA 6850

Wie der Name schon sagt, stellt der ACIA 6850 (Asynchronous Communications Interface Adapter, Baustein zur asynchronen Kommunikation) eine serielle Schnittstelle zur Verfügung. Hierfür verfügt der Baustein über zwei Register, die beim Schreiben und Lesen unterschiedliche Funktionen haben (man könnte daher auch sagen, es seien vier Register):

- CR, Control Register, Programmier-Register
- SR, Status Register, Status-Register
- SR, Send Register, Sende-Register
- RR, Receive Register, Empfangsregister

Control Register
In dieses Register kann nur geschrieben werden. Die Bits müssen entsprechend der gewünschten Funktion gesetzt werden und haben folgende Bedeutung:

Bit 0 u.1: Baud-Rate und Master Reset

Bit: 1 0	**Bedeutung**
0 0	ohne Teilung
0 1	Teilung durch 16 (Midi)
1 0	Teilung durch 64 (Tastatur)
1 1	Master Reset

Bit 2 bis 4: Paritäts-, Daten- und Stopbits

Bit: 4 3 2	**Bedeutung**
0 0 0	7 Datenbits, 2 Stopbits, gerade Parität
0 0 1	7 Datenbits, 2 Stopbits, ungerade Parität
0 1 0	7 Datenbits, 1 Stopbit, gerade Parität
0 1 1	7 Datenbits, 1 Stopbit, ungerade Parität
1 0 0	8 Datenbits, 2 Stopbits, keine Parität
1 0 1	8 Datenbits, 1 Stopbit, keine Parität
1 1 0	8 Datenbits, 1 Stopbit, gerade Parität
1 1 1	8 Datenbits, 1 Stopbit, ungerade Parität

Bit 5 u. 6: Request to Send, Break, Sender-Interrupt

Bit: 6 5	Bedeutung
0 0	RTS-Signal Low, kein Interrupt Request
0 1	RTS-Signal Low, Interrupt erlaubt
1 0	RTS-Signal High, kein Interrupt Request
1 1	RTS-Signal High, Interrupt erlaubt

Bit 7:Empfänger-Interrupt ist erlaubt, wenn der Puffer voll ist oder Überlauf aufgetreten ist.

Status Register
Dieses Register kann nur gelesen werden. Die Bits in diesem Register haben folgende Bedeutung:

Bit 0: der Empfangspuffer ist voll.
Bit 1: der Sendepuffer ist leer.
Bit 2: 1, wenn das Signal Data Carrier anliegt.
Bit 3: entspricht der CTS-Leitung (Clear to Send).
Bit 4: Framing Error, Anzahl Stopbits stimmt nicht.
Bit 5: Overrun Error, Überlauffehler
Bit 6: Parity Error, Paritäts-Fehler.
Bit 7: Interrupt Request, Bit geht bei Interrupt auf 1

In das Send Register (Sende-Register) werden die Daten hineingeschrieben, die über die serielle Schnittstelle geschickt werden sollen und im Receive Register (Empfangs-Register) können die empfangenen Daten abgeholt werden.

4.6 Der Sound-Prozessor YM 2149

Zur Musik- bzw. Geräuscherzeugung hat der Atari ST den Sound-Prozessor YM 2149 von Yamaha oder den AY-3-8910 von General Instruments, die baugleich sind.

Der Baustein verfügt über folgende Möglichkeiten:

- drei unabhängig voneinander programmierbare Ton-Generatoren
- für jeden Tonkanal eine Lautstärkenregelung
- einen programmierbaren Rauschgenerator
- einen programmierbaren Mischer für Ton und Rauschen
- eine programmierbare Hüllkurve für alle Kanäle
- zwei parallele, bidirektionale I/O-Kanäle

Zur Programmierung hat der Baustein 16 Register, die über zwei Adressen angesprochen werden. Um eines der 16 Register mit einem bestimmten Wert zu laden, muß zunächst die Nummer dieses Registers auf die erste Adresse geschrieben werden. Danach kann der eigentliche Befehl für dieses Register (das Kommando-Byte) auf die zweite Adresse geschrieben werden.

$FF8800 Select/Read, Auswahl/Lesen
Auswahl der Register-Nummer. Durch eine Leseoperation auf diese Adresse kann der aktuelle Wert des angewählten Registers gelesen werden.

$FF8802 Write/Schreiben
Das angewählte Register wird mit einem Kommando-Byte beschrieben

Die elf Register des 2149:

Register	Funktion
0	Kanal A, Feinabstimmung der Periodendauer
1	Kanal A, Grobabstimmung der Periodendauer
2	Kanal B, Feinabstimmung der Periodendauer
3	Kanal B, Grobabstimmung der Periodendauer
4	Kanal C, Feinabstimmung der Periodendauer
5	Kanal C, Grobabstimmung der Periodendauer
6	Rauschgenerator, Tonhöhe
7	Kontrollregister Kanalmischung und I/O-Ports
8	Kanal A, Lautstärke der Hüllkurve
9	Kanal B, Lautstärke der Hüllkurve
10	Kanal C, Lautstärke der Hüllkurve
11	Hüllkurve, Feinabstimmung der Periodendauer
12	Hüllkurve, Grobabstimmung der Periodendauer
13	Hüllkurve, Form
14	I/O-Port A
15	I/O-Port B

4.7 Der Parallelbaustein 8255 oder 82C55

Auf der Interface-Karte, die diesem Buch beiliegt, befindet sich in dem fest verdrahteten Teil der parallele I/O-Baustein 8255, der bei allen Anwendungen mit parallelem Datentransfer eingesetzt werden kann. Der 82C55 ist zum 8255 funktions- und pinkompatibel.

Bild 4.2 zeigt die Anschlußbelegung des 8255.

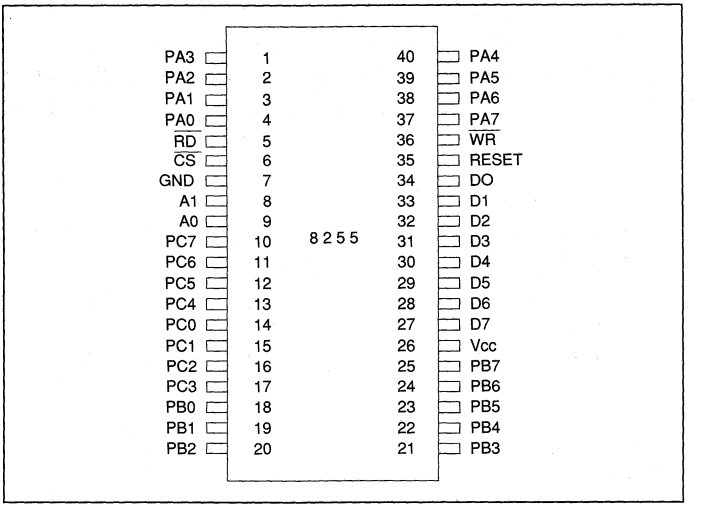


Bild 4.2: Anschlußbelegung des 8255

Der Baustein verfügt über drei unabhängige 8-Bit-Datenkanäle, die in Bild 4.3 als Kanal A, B und C bezeichnet sind. Diese Kanäle können an verschiedene Peripheriegeräte angeschlossen werden und werden vom Computer über drei Ports angesprochen. Die einzelnen Kanäle sind teilweise unabhängig voneinander in verschiedenen Arbeitsweisen zu betreiben. Neben diesen drei Datenkanälen verfügt der 8255 noch über einen Kontrollport, über den die Programmierung des Bausteins erfolgt.

Um diese vier Register ansprechen zu können, benötigt der 8255 zwei Adreßleitungen, die in Bild 4.2 mit A0 und A1 bezeichnet sind. Auf der Interface-Karte werden diese beiden Leitungen mit den Adreßleitungen A13 und A14 des Atari-Adreßbus verbunden:

8255	Atari ST
A0	A13
A1	A14

Zum Ansprechen des Bausteins werden neben den Datenleitungen noch die Anschlüsse WR (Write, Schreiben), RD (Read, Lesen), CS (Chip Select, Baustein ansprechen) und Reset (zurücksetzen) beschaltet, die an folgenden Adreßleitungen des Atari ST liegen:

8255	Atari ST
A1	Datenleitung 1
.	.
.	.
A8	Datenleitung 8

A9 CS
A10 "
A11 Schreiben, Lesen
A12 und Reset

Die Adressen A9 bis A12 liegen nicht einzeln an bestimmten Anschlüssen des 8255, sondern diese Leitungen kodieren die aufgeführten Funktionen. Erst hinter entsprechenden Bausteinen (74139, zwei zu vier Decoder) auf der Interface-Karte werden die Leitungen am 8255 beschaltet (Bilder 8.6, 8.9 und 8.10).

Der gesamte Datentransfer zwischen CPU und 8255 wird also über Adreßleitungen abgewickelt, der Datenbus ist nicht beteiligt.

Eine genauere Beschreibung dieser Funktionen befindet sich bei der Beschreibung der Interface-Karte in Kapitel 8.

Als Beispiel für den Prinzipaufbau eines I/O-Bausteins zeigt Bild 4.3 das Blockschaltbild des 8255. Auf der linken Seite ist der Anschluß an den Mikroprozessorbus dargestellt, auf der rechten Seite die parallelen Ein-/Ausgabe-Leitungen, auch Schnittstelle genannt.

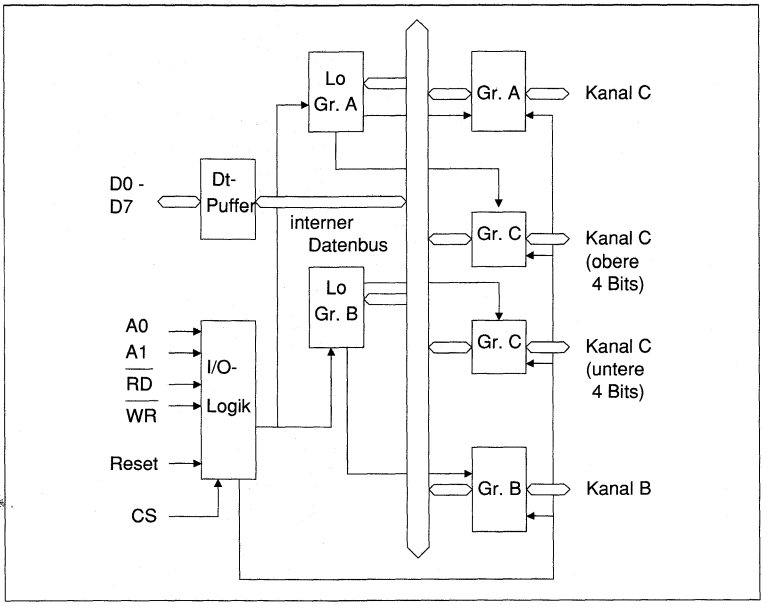


Bild 4.3: Blockschaltbild des 8255 – Ausgabe der einzelnen Kanäle

Nach einem Systemstart sind alle Kanäle des 8255 auf Eingabe geschaltet. Soll der Baustein von dieser Grundprogrammierung abweichende Funktionen erfüllen, muß die neue Arbeits-

weise zunächst durch eine Programmierung festgelegt werden. Port A, B und C können jeweils auf Ein- oder Ausgabe eingestellt werden. Port C ist nochmals in zwei 4-Bit-Ports aufzuspalten, die unabhängig voneinander auf Ein- oder Ausgabe programmierbar sind. Außerdem kann Port C im sogenannten Bit-Set-Reset-Modus betrieben werden. Hierbei wird jedes einzelne Bit (also Leitung) unabhängig von den restlichen durch Schreiboperationen auf den Kontrollport gesetzt (logisch 1) bzw. zurückgesetzt (logisch 0). Die Funktionsweise von Port C ist für eine Schnittstelle mit Handshake-Leitungen besonders wichtig. Das Programmieren des 8255 geschieht durch das Schreiben eines 8-Bit-Wortes auf den Kontrollport. Jedes einzelne Bit dieses Bytes hat eine eindeutig festgelegte Bedeutung. Es gibt zwei verschiedene Arten von Programmierwörtern, die durch das höchste Bit unterschieden werden. Bild 4.4 zeigt den Aufbau des Wortes zum Festlegen der Arbeitsweise (Mode Definition Format) des 8255, während Bild 4.7 den Aufbau des Wortes zum Setzen und Zurücksetzen (Bit Set/Reset Format) der Leitungen des Ports C darstellt.

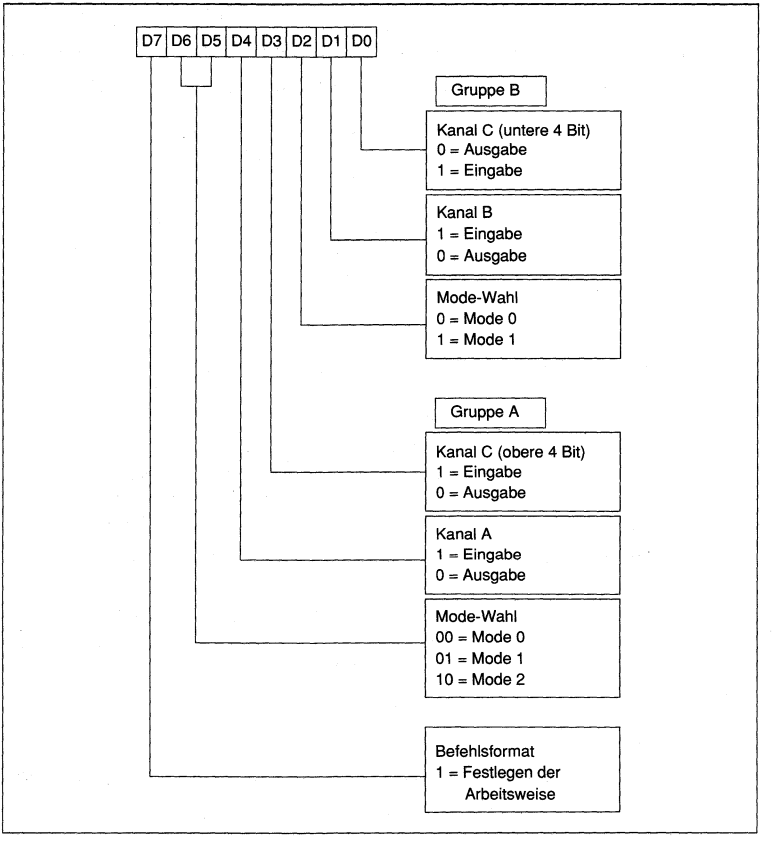


Bild 4.4: Mode-Definition-Format des 8255

Die Ports des 8255 werden in zwei Gruppen aufgeteilt. Die Kanäle der beiden Gruppen können nur zusammen programmiert werden.

Gruppe A: Kanal A und die 4 höheren Leitungen von Kanal C
Gruppe B: Kanal B und die 4 unteren Leitungen von Kanal C

Der 8255 kann in drei verschiedenen Modi betrieben werden:
Mode 0 – Einfache Ein-/Ausgabe (Basic Input/Output)
Mode 1 – Strobed Ein-/Ausgabe (Strobed Input/Output)
Mode 2 – Bidirektional

In Mode 0 wird der Datentransfer über die einzelnen Ports durch einfache Lese- und Schreiboperationen durchgeführt. Es findet keine Kontrolle statt. Alle 24 Leitungen stehen für die Daten zur Verfügung. In Mode 1 gibt es zwei 8-Bit-Ports mit jeweils drei Kontroll-Leitungen, die einen Hardware-Handshake realisieren (Strobed I/O, Quittungs-E/A). Die beiden noch verbleibenden Leitungen können für andere Zwecke verwendet werden.

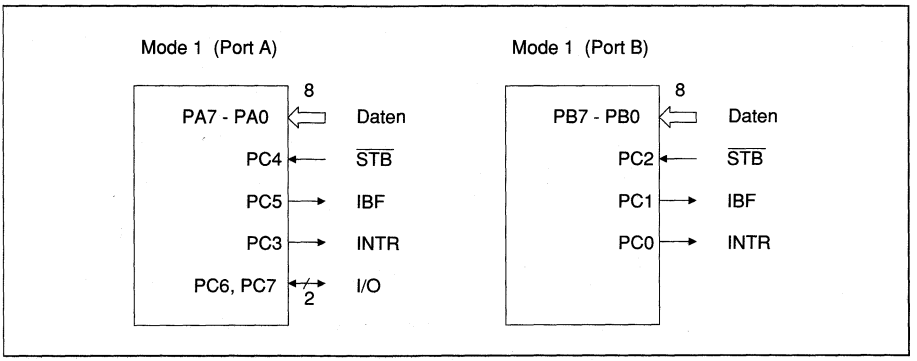


Bild 4.5a: Beispiel der Leitungen in Mode 1

Bild 4.5a zeigt als Beispiel zwei mögliche Aufteilungen des Leitungen des 8255 bei dieser Betriebsart. Hierbei haben die Kontrolleitungen folgende Bedeutung:

STB Strobe Input, Eingabefreigabe.
IBF Input Buffer Full, Eingabepuffer voll.
INTR Interrupt Request, Unterbrechung anfordern.
OBF Output Buffer Full, Ausgabpuffer voll.
ACK Acknowledge Input, Eingabequittung.

In Mode 2 kann nur die Gruppe A betrieben werden. Hierbei werden mit Port A sowohl Daten gesendet als auch empfangen. Fünf Leitungen des Ports C dienen dem Handshakebetrieb. Port B und drei Leitungen von Port C sind davon unabhängig, weiterhin zum einfachen Empfangen oder Senden von Daten zu verwenden.

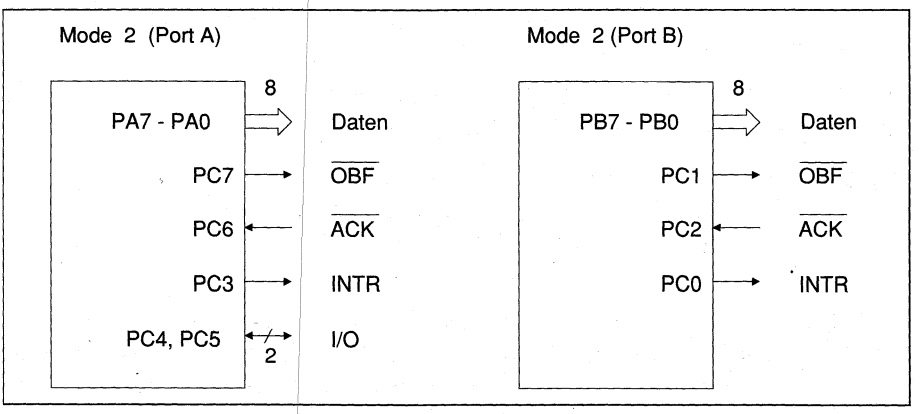


Bild 4.5b: Beispiel der Leitungen in Mode 2

Bild 4.5b zeigt diese Möglichkeit. Hierbei haben die Abkürzungen dieselbe Bedeutung wie bei Mode 1.

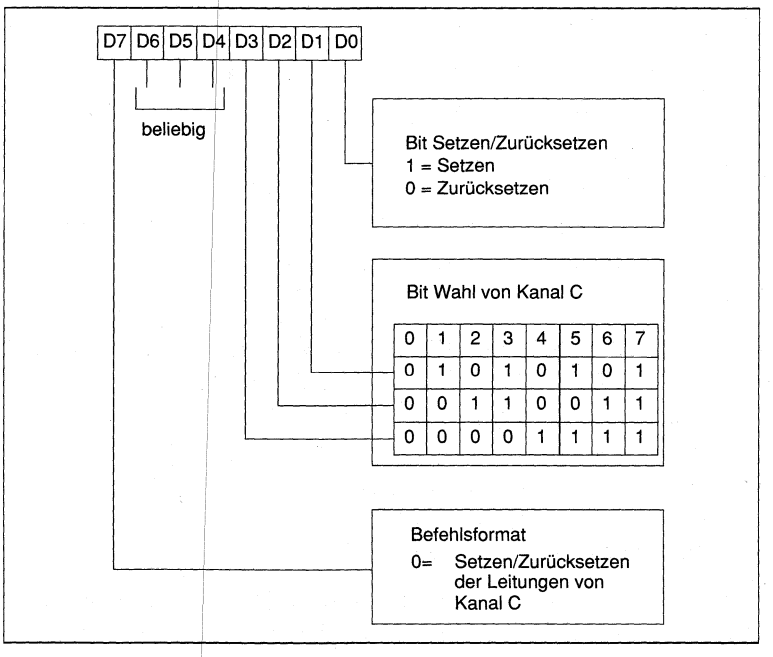


Bild 4.6: Bit-Set/Reset-Format des 8255

Beim Programmierwort zum Festlegen der Arbeitsweise des 8255 werden die drei niederwertigen Bits D0, D1 und D2 für die Gruppe B benutzt, die darauf folgenden vier Bit D3, D4, D5 und D6 für die Gruppe A. Bit D7 dient zur Unterscheidung der zwei verschiedenen Kontrollwörter:

D7 = 1 Programmieren der Arbeitsweise (Mode Definition Format)
D7 = 0 Programmieren der Leitungen von Kanal C (Bit-Set/Reset-Format).

Mit den Angaben aus Bild 4.4 kann man das Format des Kontrollworts für eine bestimmte Arbeitsweise des 8255 zusammensetzen. Zur Programmierung wird dieses Wort auf den Kontrollport des 8255 geschrieben. Einige Beispiele für Kontroll- oder Programmierwörter zum Festlegen der Arbeitsweise des 8255 sind:

1000 000080h alle Kanäle (A, B, C) auf Ausgabe
1001 10119Bh alle Kanäle (A, B, C) auf Eingabe
1001 100098h A und C höherwertig auf Eingabe
B und C niedrigerwertig auf Ausgabe

Im Bit-Set/Reset-Format können die Leitungen von Kanal C einzeln gesetzt und zurückgesetzt werden, ohne die anderen Leitungen dieses Kanals zu beeinflussen. Hierzu wird ein Programmierwort auf den Kontrollport des 8255 geschrieben, bei dem Bit D7 gelöscht ist. Durch die Bits D1 bis D3 wird eine der acht Leitungen von Kanal C ausgewählt. Um acht Leitungen zu bezeichnen, benötigt man eine dreistellige Binärzahl:

D3	D2	D1	angewählte Leitung
0	0	0	Leitung 1
0	0	1	Leitung 2
0	1	0	Leitung 3
.	.	.	usw

Durch Bit D0 wird angegeben, ob die angewählte Leitung auf logisch 1 (High) oder logisch 0 (Low) gesetzt werden soll. Bit D4 bis D6 haben bei dieser Programmierart keine Bedeutung. Durch eine solche Schreiboperation bleiben alle anderen Leitungen von Kanal C unbeeinflußt. Einige Beispiele für das Programmieren der Leitungen von Kanal C sind:

0000 0111 07h Leitung 4 von Kanal C gesetzt (High)
0000 1110 0Eh Leitung 8 von Kanal C zurückgesetzt (Low)

Zum Abschluß sollen die wesentlichen Eigenschaften des 8255 noch einmal zusammengefaßt werden:

1. Der 8255 ist eine paralleler I/O-Baustein mit drei unabhängigen Ein- und Ausgabekanälen, die in verschiedenen Betriebsarten verwendet werden können.
2. Nach dem Reset sind alle 24 Leitungen auf Eingabe (Input) geschaltet.

3. Durch eine Schreiboperation auf den Kontrollport wird die gewünschte Arbeitsweise angewählt.
4. Über die Kanäle A, B und C können nach dem Programmieren Daten geschrieben oder gelesen werden.
5. Das Setzen oder Zurücksetzen von Bits am Kanal C geschieht durch ein Programmierwort auf den Kontrollport.

4.8 Der serielle Baustein 8251

Der zweite hochintegrierte Baustein auf der Interface-Platine ist der 8251, ein Mikrocontroller zur seriellen Datenübertragung. Bild 4.7 zeigt die Anschlußbelegung dieses Bausteins.

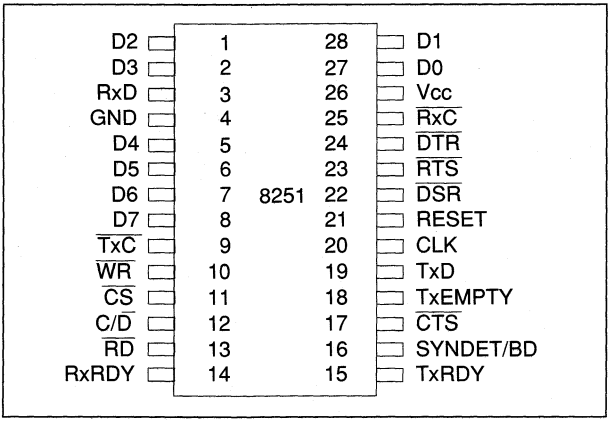


Bild 4.7: Anschlußbelegung des 8251

Der 8251 beinhaltet intern alle notwendigen Funktionen, um eine serielle Schnittstelle nach V24 zu bedienen. Um die Einzelheiten der Übertragung wie Start-, Stop- und Paritätsbit sowie die Baud-Rate braucht sich der Benutzer nicht kümmern. Nachdem diese Werte durch eine Programmierung einmal eingestellt sind, überträgt der Baustein die Daten richtig formatiert an den Empfänger bzw. setzt die empfangenen Daten wieder zu einem Datenbyte zusammen.

Bild 4.8 zeigt das Blockdiagramm des internen Aufbaus dieses Bausteins sowie die auf der Computer- und Schnittstellenseite benötigten Anschlüsse. Der 8251 verfügt über einen Full-Duplex-Übertragungskanal, der von dem Computer über zwei Port-Adressen, einem Daten- und einem Kontroll- bzw. Status-Port, angesprochen wird. Die Anschlüsse für den Systembus sind in Bild 4.8 links oben eingezeichnet. Sie entsprechen denen beim 8255. Da hier nur zwei I/O-Ports benötigt werden, kommt der Baustein mit einer Adreßleitung aus. Alle

anderen Adreßleitungen des Systembusses müssen wieder von einer externen Logik dekodiert werden. Die Leitungen links unten dienen zum Anschluß eines Modems (ein Gerät mit dem Daten über das Telefonnetz übertragen werden). Auf der rechten Seite sind die Leitungen für die serielle Schnittstelle dargestellt. Die wichtigsten Leitungen sind:

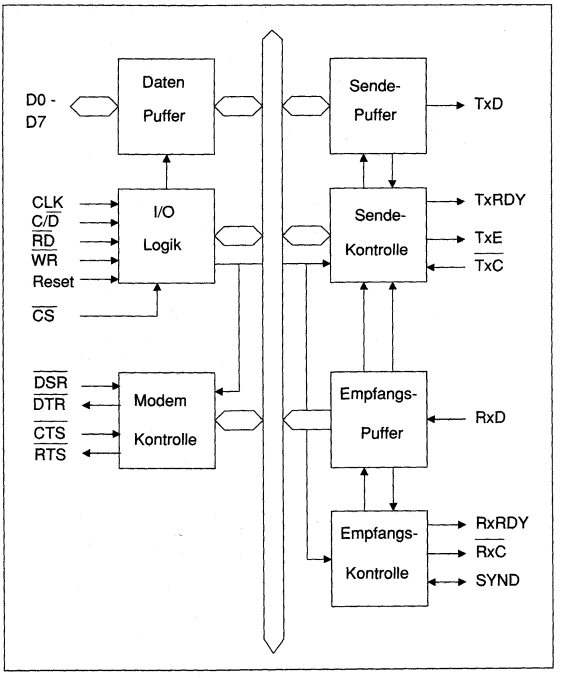


Bild 4.8: Blockschaltbild des 8251

TxD Transmit Data – Daten übertragen
RxD Receive Data – Daten empfangen
TxC Transmitter Clock – Übertragungsrate

Die Daten werden mit dieser Frequenz oder einem Bruchteil davon übertragen

RxC Receiver Clock – Empfängerrate
TxRDY Transmitter Ready – Sender bereit

Ist dieses Signal aktiv, so kann von der CPU ein weiteres Datenbyte zur Übertragung geschickt werden

RxRDY Receiver Ready – Empfänger bereit

Ist dieses Signal aktiv, so kann ein Datenbyte von der CPU abgeholt werden

Der 8251 belegt zwei hintereinanderliegende Port-Adressen:

I/O-Adresse	Bezeichnung beim 8251
60h	Datenport (senden und empfangen)
61h	Kontroll- und Statusport

Anders als der 8255 muß der Baustein erst programmiert werden, bevor er Daten übertragen kann. Nach dem RESET erwartet der 8251 ein Programmierwort zum Festlegen der gewünschten Arbeitsweise (Mode Instruction), danach können Daten oder ein Kommandowort (Command Instruction) gesendet werden (Bild 4.9).

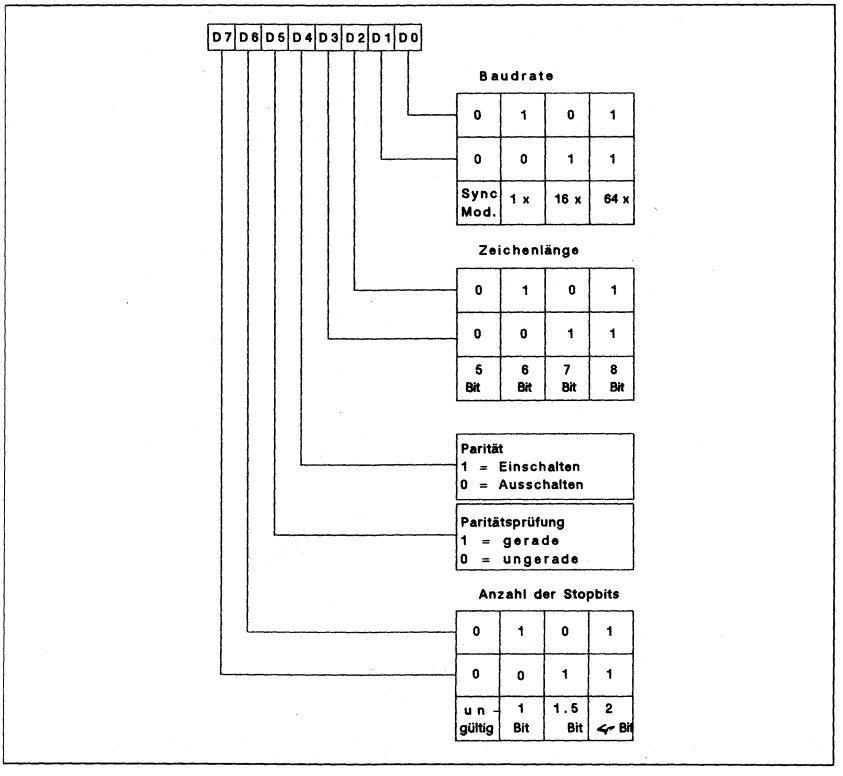


Bild 4.9: Mode-Instruction-Format des 8251

Bild 4.9 zeigt das Format des Programmierworts zum Festlegen der Arbeitsweise des 8251. Ein solches Wort darf nur nach einem Reset übergeben werden. Alle danach an den Kontrollport übergebenen Datenbytes werden als Kommandowörter interpretiert. Bit D0 und D1 legen zusammen mit dem an den Baustein angelegten Oszillator die Übertragungsrate fest. Ein solcher Oszillator wird benötigt, da die Daten in einem festen Takt übertragen werden.

Steht in D0 und D1 eine 0, so arbeitet der 8251 im Synchronmodus, in dem Bit D2 bis D7 eine teilweise andere Bedeutung haben als in Bild 4.10. Bit D2 und D3 bestimmen die Zeichenlänge bei der Übertragung. D4 schaltet die Paritätserzeugung und -prüfung ein bzw. aus. Falls mit Parität gearbeitet wird, bestimmt Bit D5 die Art der Parität. Das Paritätsbit wird bei gerader Parität auf 1 gesetzt, wenn in dem übertragenen Datenbyte eine gerade Anzahl von Einsen enthalten ist, sonst auf Null. Auf diese Weise kann festgestellt werden, ob bei der Übertragung ein Ein-Bit-Fehler aufgetreten ist, da sich die Parität dabei ändert. Bit D6 und D7 bestimmen die Anzahl der Stop-Bits bei der Übertragung.

Der aktuelle Zustand der Datenübertragung kann durch Lesen des Kontrollports jederzeit abgefragt werden. Bei einer solchen Operation wird vom 8251 ein Datenbyte mit dem in Bild 4.11 dargestellten Format ausgegeben. Damit kann z.B. festgestellt werden, ob ein Datenbyte bereits abgeschickt worden ist oder empfangen wurde.

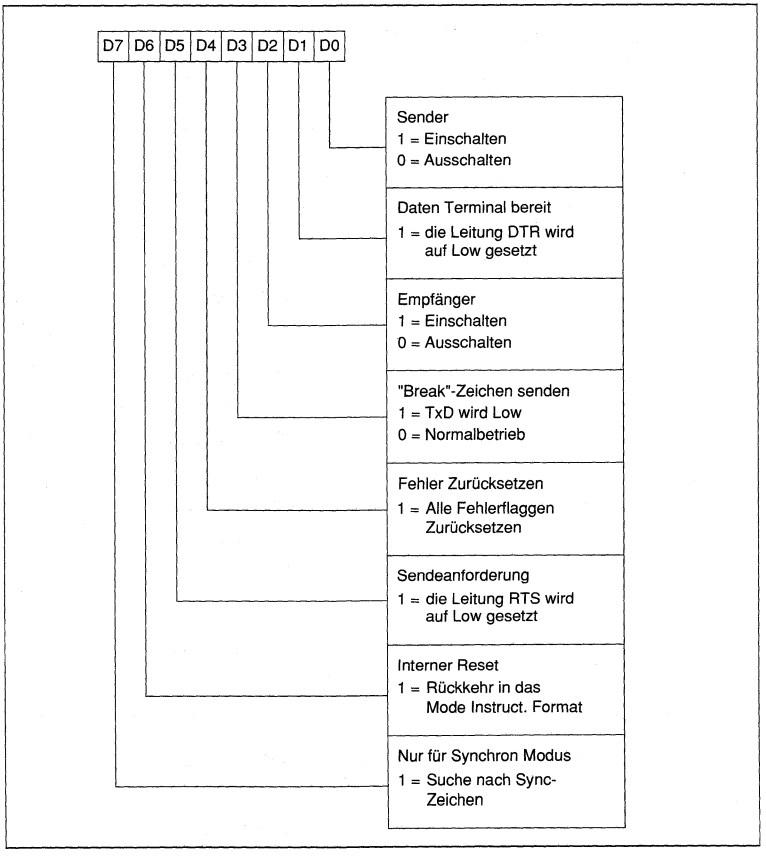


Bild 4.10: Command-Instruction-Format des 8251

Besonders wichtig sind hierbei die Bits D0 TxRDY (Transmitter Ready – Sender bereit) und D1 (Receiver Ready – Empfänger bereit). Befindet sich in D0 eine 0, so ist der Sendepuffer leer, also das letzte Zeichen abgeschickt. Die CPU kann dem 8251 ein neues Datenbyte für die Übertragung zur Verfügung stellen. Wird von der CPU ein Datenbyte geschickt, bevor der Sendepuffer leer ist, setzt der Baustein im Statusbyte ein Fehlerbit, was die Übertragung aber nicht stört. Allerdings geht ein Datenbyte verloren. Bei einer 0 in Bit D1 steht im Empfangspuffer des 8251 ein Datenbyte zur Verfügung. Dieses muß von der CPU abgeholt werden, bevor das nächste Byte von der Übertragungsleitung eintrifft. Geschieht dies nicht, wird auch hier ein Fehlerbit gesetzt, ohne die weitere Funktion zu stören. Wie im ersten Fall geht auch hier das Datenbyte verloren. Bit D0 bis D2 und D6 haben dieselbe Bedeutung wie die entsprechenden Anschlüsse des 8251. Bit D3 bis D5 sind Fehlerflaggen, die beim Auftreten des entsprechenden Fehlers gesetzt werden.

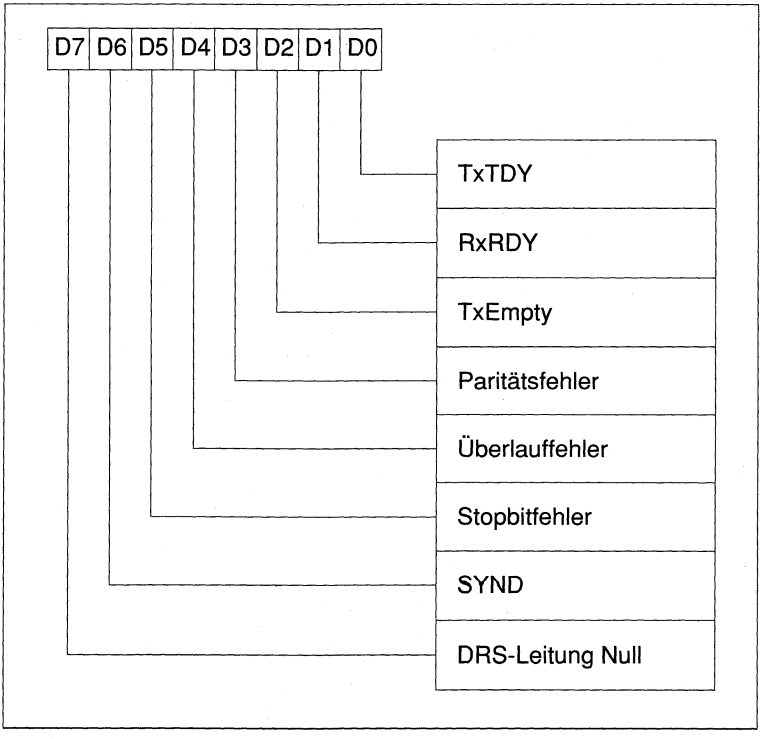


Bild 4.11: Status-Read-Format des 8251

4.9 Speicherbausteine

In allen Computern werden zwei verschiedene Arten von Speicherbausteinen benutzt:

- ROM (Read Only Memory) – Festwert- oder Nur-Lese-Speicher
- RAM (Random Access Memory) – Schreib-/Lese-Speicher

Diese beiden Arten werden auch im Atari ST verwendet.

- ROM (Read Only Memory) – Festwert- oder Nur-Lese-Speicher

Diese Speicherelemente können im normalen Betrieb nur gelesen, aber nicht beschrieben und somit verändert werden. Jeder Mikrocomputer benötigt in einem Teil seines Adreßraums einen solchen Speicher, da der Computer beim Einschalten auf einer bestimmten Adresse den ersten Befehl erwartet. Im allgemeinen wird dieses Urprogramm dazu benutzt, das eigentliche Betriebssystem von einem externen Speichermedium, z.B. Diskette oder Festplatte zu laden. Auch Teile des Betriebssystems eines Rechners, das sogenannte BIOS (Basic Input Output System) sind in einem ROM untergebracht.

Liste der gebräuchlichen ROM-Typen:

ROM	Read Only Memory	Festwertspeicher
PROM	Programmable ROM	Programmierbares ROM
EPROM	Erasable PROM	Löschbares PROM
EEPROM	Electric EPROM	Elektrisches EPROM

Die Bezeichnung ROM gilt streng genommen nur für solche Festwertspeicher, die schon bei der Herstellung alle Informationen enthalten. In der Regel ist dieser Prozeß sehr aufwendig und lohnt sich nur bei großen Stückzahlen. PROMs dagegen können nachträglich durch ein spezielles Gerät programmiert werden. EPROMs sind ebenfalls programmierbare ROMs, die im Gegensatz zu den PROMs aber wieder gelöscht werden können. Hier sind heute zwei verschiedene Typen gebräuchlich. Das normale EPROM kann mit UV-Licht gelöscht werden. Zu diesem Zweck ist auf dem Baustein ein kleines Quarzglasfenster eingelassen, unter dem der eigentliche Chip auch gut zu erkennen ist. Nach wenigen Minuten unter einer UV-Lampe ist die Information gelöscht und der Baustein kann neu programmiert werden. Eine Zusatzkarte zum Programmieren dieser Bausteine ist mittlerweile auch für den Atari ST zu bekommen. Beim EEPROM wird die Information auf elektrischem Weg gelöscht.

- RAM (Random Access Memory) – Schreib-Lese-Speicher

In diese Speicher können Informationen geschrieben und wieder ausgelesen werden. Alle Programme und Daten von einer Diskette oder Festplatte werden im RAM abgelegt, bevor sie von der Zentraleinheit weiterverarbeitet werden. Bei den Schreib-Lese-Speicherbausteinen kann man zwischen zwei verschiedenen Typen unterscheiden:

- statischer Speicher
- dynamischer Speicher

Diese beiden Arten unterscheiden sich im internen Aufbau der einzelnen Speicherzellen. Beim statischen Speicher bleiben die Daten erhalten, solange eine Versorgungsspannung anliegt. Beim dynamischen Speicher gehen alle Informationen nach wenigen Millisekunden verloren, wenn sie nicht ständig aufgefrischt werden. Im Atari ST wird der Refresh von der MMU durchgeführt. Die Speicherbausteine unterscheiden sich auch noch hinsichtlich ihrer internen Organisation und der Speicherkapazität. Es gibt byteweise und bitweise organisierte Speicher. Bausteine, die byteweise organisiert sind, benötigen für die Daten acht Anschlußleitungen. Bei der bitweisen Organisation werden acht Bausteine zum Speichern eines Byte und 16 für ein Wort benötigt. Jeder Speicherbaustein verfügt neben den Adreßleitungen über nur eine Datenleitung. Beim Atari ST sind maximal zwei Bänke mit jeweils 16 Speicherbausteine vorgesehen (siehe Kapitel 5).

5 Aufbau und Funktionsweise des Atari ST

5.1 Hardwareüberblick

In den ersten Kapiteln dieses Buches wurden die allgemeinen Prinzipien vorgestellt, nach denen Computer arbeiten. In diesem Kapitel soll nun speziell der Aufbau und die Funktionsweise des Atari ST beschrieben werden. Bild 5.1 zeigt in einem Blockdiagramm alle Funktionsgruppen der Systemplatine des Atari ST im Überblick.

Wie beim Menschen das Rückgrat alle Teile des Körpers mit dem Gehirn verbindet, verhält es sich bei einem Computer mit dem Systembus. Über seine Leitungen werden alle Informationen innerhalb des Computers ausgetauscht. Das Gehirn eines Computers ist die CPU, beim Atari ST die Zentraleinheit 68000. Die CPU steuert alle Busaktivitäten. Gleichwohl können bestimmte Bausteine ihre Funktionen auch unabhängig von der Zentraleinheit ausführen, ja sogar für einen bestimmten Zeitraum die Kontrolle über den Bus übernehmen (siehe auch Kapitel 4.2).

Beim Atari ST umfaßt der Adreßbus 23 (A1 bis A22) und der Datenbus 16 Leitungen. Durch das Fehlen der Leitung A0 kann der Prozessor mit Hilfe der Adreßleitungen nur auf Speicherworte (16 Bit) zugreifen, die bei einer geraden Adresse beginnen (0, 2, 4, ...) Da es bei vielen Anwendungen aber auch notwendig ist, den Speicher byteweise zu adressieren, hat der Prozessor zwei Steuerleitungen UDS (Upper Data Strobe, oberer Datentakt, aktiv Low) und LDS (Lower Data Strobe, unterer Datentakt, aktiv Low), die dies ermöglichen. Diese beiden Leitungen gehören zum Kontrollbus, der alle Leitungen zur Verwaltung (oder Kontrolle) des Gesamtsystems enthält, z.B. die Leitungen für den DMA-Betrieb, zur Organisation der Interrupts, zur Unterscheidung zwischen Schreib- und Leseoperationen beim Speicher usw. Bild 5.1 zeigt ein Blockschaltbild vom Aufbau des Atari ST. Im Zentrum dieses Bildes befindet sich der Systembus, der neben den bisher beschriebenen Leitungen noch weitere Kontrollsignale enthält.

Die wichtigen Aufgaben zur internen Systemverwaltung werden neben der CPU (68000) von den Custom Chips (kundenspezifische Bausteine, siehe auch Kapitel 2.4) übernommen, die im Folgenden kurz vorgestellt werden. Die Kommunikation mit der Peripherie, z.B. der Tastatur, dem Diskettenlaufwerk, dem Drucker, der MIDI-Schnittstelle, erfolgt durch die in Kapitel 4 beschriebenen Bausteine. Wobei auch hier die Custom Chips eine Rolle spielen.

Manche Funktionen werden von mehreren Bausteinen gemeinsam verwaltet. Es ist also nicht immer möglich, eine bestimmte Funktion eindeutig einem Baustein zuzuordnen. Bei den kundenspezifischen Chips ist dies völlig unmöglich, da sie in enger Wechselwirkung stehen und sich in ihren Funktionen gegenseitig ergänzen.

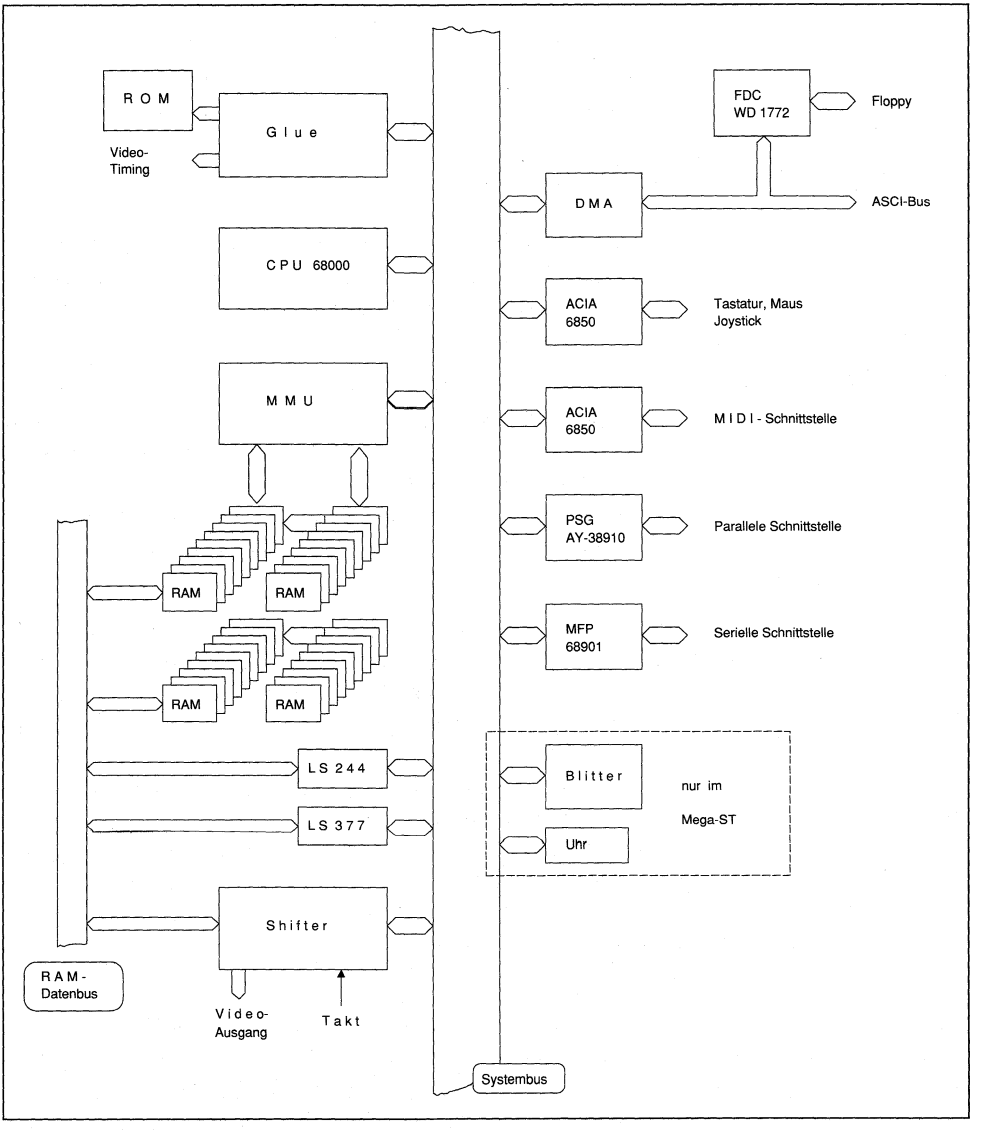


Bild 5.1: Blockschaltbild der Systemplatine des Atari ST

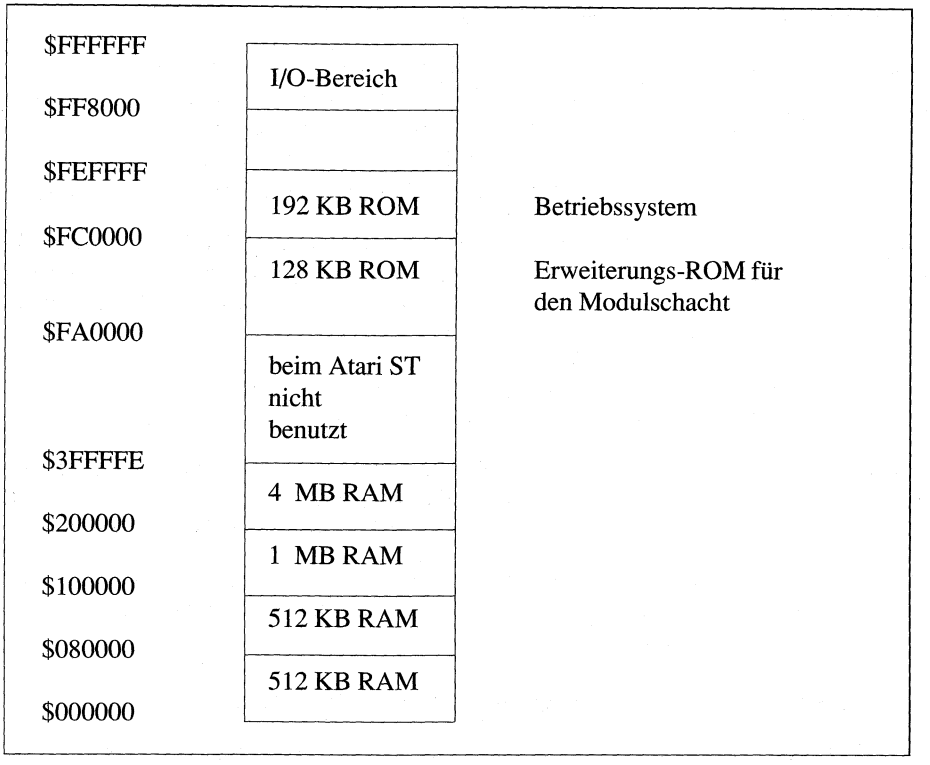


Bild 5.2: Aufteilung des Gesamtspeichers beim Atari ST

Die Zentraleinheit 68000 hat 23 Adreßleitungen und kann demnach 16 Megabyte (16 Mbyte) Speicher verwalten. Wie dieser Gesamtspeicher im Atari ST aufgeteilt ist, zeigt Bild 5.2. Bei 23 Adreßleitungen benötigt man eine sechsstellige hexadezimale Zahl, um jede Speicherstelle zu bezeichnen. Im Atari ST werden nur die unteren 4 Mbyte des Adreßbereichs ($000000 bis $3FFFFF) als Schreib-/Lesespeicher (RAM, Random Access Memory, Speicher mit wahlfreiem Zugriff) genutzt (mehr ist mit 22 Adreßleitungen nicht möglich). Damit kann der Atari ST bis auf 4 MByte aufgerüstet werden. Wie oben beschrieben, ist dies auch der maximale Speicher, den die MMU verwalten kann.

Der Adreßbereich von $FA0000 bis $FEFFFF ist für das ROM reserviert. Das Betriebssystem liegt im oberen Teil dieses Speichers, während die Adressen von $FA0000 bis $FBFFFF für ROM-Erweiterungen vorgesehen sind. Am oberen Ende des Speichers von $FF8000 bis $FFFFFF befindet sich der I/O-Bereich. Um die RAM-, ROM- und I/O-Bereiche voneinander zu unterscheiden, benötigt man eine Logik, in der die Adressen dekodiert und die richtigen Bausteine angesprochen werden, d.h. mit einem Chip-Select-Signal versorgt werden. Diese Logik befindet sich in einem der Custom-Chips, dem Glue.

Glue (Leim, Kleister, aber auch leimen)

Dieser Chip – nomen est omen – ist der Leim zwischen den verschiedenen Bausteinen (Funktionsgruppen) des Atari ST. Im Glue wird die vollständige Adreßdekodierung für das RAM und den I/O-Bereich vorgenommen, weswegen hier der gesamte Adreßbus anliegt (A1 bis A23). Der Glue erzeugt direkt die CS-Signale (Chip Select, Baustein ansprechen) für die Peripheriebausteine. Auch das VPA-Signal (Valid Peripheral Adress) wird von Glue erzeugt.

Versucht der Prozessor auf einen nicht vorhandenen Speicherbereich oder im User Mode auf einen Peripheriebaustein zuzugreifen, so löst der Glue einen Busfehler aus (BERR-Signal, führt zum Interruptvektor Nr. 2). Daher sind auch die drei Funktionscodeleitungen des 68000 FC0 bis FC2 an dem Glue angeschlossen.

Auch die Interrupts werden vom Glue verwaltet. Deshalb sind Interruptleitungen IPL1 und IPL2 hier angeschlossen. Die Leitung IPL0 wird bei Atari ST nicht benutzt.

Am Glue liegt der Systemtakt von 8 MHz an, woraus alle anderen benötigten Frequenzen gewonnen werden: 2 MHz für den Sound-Chip, 500 kHz für den Tastaturtakt und die MIDI-Schnittstelle, sowie die Signale HSYNC, VSYNC, BLANK und DE für den Monitor.

Der DMA-Betrieb wird ebenso vom Glue verwaltet wie die Interrupts. Natürlich führt der Glue diese Funktionen nicht alleine aus, sondern im engen Wechselspiel mit allen anderen Bausteinen der Systemplatine.

MMU (Memory Manager Unit, Speicherverwaltungs-Einheit)

Wie der Name schon sagt, verwaltet die MMU den gesamten Speicher des Atari ST. Hierfür ist die MMU mit den Adreßleitungen A1 bis A21 verbunden, was den oben erwähnten Adreßbereich von 4 Mbyte ergibt (im Atari ST werden also nicht alle Adreßleitungen des 68000 verwendet).

Die wichtigste Aufgabe der MMU besteht in der Verbindung der dynamischen RAM-Bausteine mit dem Bus des Prozessors. Die dynamischen RAM-Chips haben einen gemultiplexten Adreßbus, d.h. an diese Bausteine kann nicht die ganze Adresse angelegt werden, sondern es muß zunächst die Zeilen- und dann die Spaltenadresse angelegt werden. Hierfür sind zwei Steuer-Leitungen nötig: RAS (Row Adress Strobe, Zeilenadresse) und CAS (Column Adress Strobe, Spaltenadresse). Die MMU stellt alle diese Signale für beide Speicherbänke zur Verfügung.

Da der Shifter ebenso wie der DMA-Chip über keine eigenen Adreßleitungen verfügt, übernimmt die MMU für diese beide Bausteine auch die Adressierung des Speichers. Hierfür übergeben diese Chips der MMU die Startadresse, die dann bei jedem Wort weitergezählt wird. Im Fall des Shifters wird ca. alle 500 ns ein Wort aus dem Video-Ram gelesen und an diesen geliefert.

Beim DMA-Betrieb geht der Datentransfer zwischen dem Speicher und dem DMA-Chip nach demselben Prinzip, hier sind aber beide Richtungen möglich.

An der Adresse $FF8001 ist die Speicherkonfiguration des Atari ST abgelegt. Hierbei haben nur die Bits 0 bis 3 eine Bedeutung. Bit 0 und 1 zeigen die Konfiguration von Bank 0, Bit 2 und 3 die von Bank 1.

Shifter

Der Shifter erzeugt aus den im Video-RAM vorhandenen Bytes das Monitorbild. Hierfür wird das RAM in regelmäßigen Zeitabständen ausgelesen und die einzelnen Bits als Pixel auf dem Bildschirm dargestellt. Das Video-RAM ist hierbei ein Teil des normalen Hauptspeichers des Atari ST. Wie bei der MMU beschrieben, kann der Shifter den Speicher nicht selbst adressieren. Diese Aufgabe wird von der MMU übernommen.

Wo sich dieser Speicher befindet, steht in den Adressen $FF8201 und $FF8203. Die unteren 8 Bit der Startadresse des Video-Rams sind immer Null. Damit kann seine Lage nur in Schritten von 256 Byte verschoben werden. Für das Video-Ram werden vom System 32 Kbyte reserviert.

Beim Monochrombetrieb ist jedem Bit im Speicher ein Bildpunkt zugeordnet. Hierbei gehört der Punkt der oberen linken Ecke zu Bit 15 des ersten Worts im Video-RAM, der zweite Punkt gehört zu Bit 14 usw. Bei einer Auflösung von 640 x 400 werden also 32000 Byte benötigt.

Die wesentliche Aufgabe des Shifters liegt darin, alle für den Monitor notwendigen Signale zu erzeugen.

DMA (Direct Memory Access, Direkter Speicherzugriff)

Der DMA-Chip dient zum Datentransfer zwischen dem Hauptspeicher und Peripheriegeräten, wie z.B. einem Diskettenlaufwerk, einer Festplatte, einem Laser-Drucker, einem Netzwerk oder ähnlichen Geräten.

Die DMA-Schnittstelle ist über acht Leitungen mit dem DMA-Chip verbunden, während am DMA-Chip die 16 Bit vom Datenbus des Atari-PC liegen.

Bei einer DMA-Operation wird dem DMA-Baustein die Basisadresse, ab der die Daten gelesen oder geschrieben werden sollen und die Anzahl der zu transferierenden Daten (in Einheiten von 512 Byte) mitgeteilt. Wie bei der MMU beschrieben, wird der Speicher nicht von dem DMA-Baustein adressiert, sondern von der MMU.

Normalerweise wird der DMA-Kanal beim Atari ST für die Disketten- und Festplattenzugriffe verwendet.

5.2 Der Input-/Output-Bereich

Im Gegensatz zu anderen Mikroprozessoren, bei denen für Ein- und Ausgabeoperationen spezielle Steuerleitungen und Befehle zur Verfügung stehen, benutzt der 68000 hierfür ganz normale Speicherplätze (dieses Konzept wurde in Kapitel 4.2 erläutert). Wie in Bild 5.2 gezeigt, liegt der für I/O-Operationen reservierte Speicher am oberen Ende des Adreßbereichs von $FF8000 bis $FFFFFF. Bild 5.3 zeigt die Zuordnung dieses Bereichs zu den einzelnen Funktionsgruppen.

$FFFC20	Uhr
$FFFC00	ACIA 6850
$FFFA40	MC 68881 Corozessor
$FFFA00	MFP 68901
$FF8A00	Blitter
$FF8800	Sound-Chip YM 2149
$FF8600	DMA / WD 1772
$FF8400	reserviert
$FF8200	Video-Baustein
$FF8000	Speicherausbau

Bild 5.3: Aufteilung des I/O-Bereichs beim Atari ST

In der folgenden Tabelle sind noch einmal alle bekannten I/O-Adressen beim Atari ST zusammengestellt (in Hex):

Adresse	Art	Funktion
	MMU	
$FF8001	R/W	Speicherkonfiguration
	Shifter	
$FF8201	R/W	Video-RAM Highbyte
$FF8203	R/W	Video-RAM Midbyte
$FF8205	R	Videoaddress Counter High
$FF8207	R	Videoaddress Counter Mid
$FF8209	R	Videoaddress Counter Low
$FF820A	R/W	Sync Modus
$FF8240	R/W	Farbpalettenregister 0
$FF8242	R/W	Farbpalettenregister 1
.		.
.		.
$FF825C	R/W	Farbpalettenregister 14
$FF825E	R/W	Farbpalettenregister 15
	DMA-Prozessor	
$FF8604	R/W	Zugriff auf das FDC-Register bzw. ACSI-Bus
$FF8604	R/W	Zähler für die Datenblöcke
$FF8606	R	Status für die DMA-Einheit
$FF8606	W	Kommando für die DMA-Einheit
$FF8609	R/W	DMA Counter High
$FF860B	R/W	DMA Counter Mid
$FF860D	R/W	DMA Counter Low
	YM-2149	
$FF8800	R	Daten lesen
$FF8800	W	Register selektieren
$FF8802	W	Daten schreiben
	Blitter	
$FF8A00	R/W	Halbton RAM 0
$FF8A02	R/W	Halbton RAM 1
.		.
.		.
$FF8A1C	R/W	Halbton RAM 14
$FF8A1E	R/W	Halbton RAM 15
$FF8A20	R/W	Source X Increment

Adresse	Art	Funktion
$FF8A22	R/W	Source Y Increment
$FF8A24	R/W	Source High Word
$FF8A26	R/W	Source Low Word
$FF8A28	R/W	Endmask 1
$FF8A2A	R/W	Endmask 2
$FF8A2C	R/W	Endmask 3
$FF8A2E	R/W	Destination X Increment
$FF8A30	R/W	Destination Y Increment
$FF8A32	R/W	Destination Address High
$FF8A34	R/W	Destination Address Low
$FF8A36	R/W	X-Count
$FF8A38	R/W	Y-Count
$FF8A3A	R/W	Half Tone Operation, Halbton-Operation
$FF8A3B	R/W	Logic Operation, logische Operation
$FF8A3C	R/W	Line Number, SMUDGE, HOG, Busy
$FF8A3D	R/W	SKEW, NFSR, FXSR
	MFP 68901	
$FFFA01	R	Datenregister des Parallelports
$FFFA03	R/W	Active Edge Register
$FFFA05	R/W	Data Direction Register, Datenrichtungsregister
$FFFA07	R/W	Interrupt Enable Register A
$FFFA09	R/W	Interrupt Enable Register B
$FFFA0B	R/W	Interrupt Pending Register A
$FFFA0D	R/W	Interrupt Pending Register B
$FFFA0F	R/W	Interrupt In Service Register A
$FFFA11	R/W	Interrupt In Service Register B
$FFFA13	R/W	Interrupt Mask Register A
$FFFA15	R/W	Interrupt Mask Register B
$FFFA15	R/W	Vektor Register
$FFFA19	R/W	Timer A Control Register
$FFFA1B	R/W	Timer B Control Register
$FFFA1D	R/W	Timer C and D Control Register
$FFFA1F	R/W	Timer A Data Register
$FFFA21	R/W	Timer B Data Register
$FFFA23	R/W	Timer C Data Register
$FFFA25	R/W	Timer D Data Register
$FFFA27	R/W	Synchronous Character Register
$FFFA29	R/W	USART Control Register
$FFFA2B	R/W	Receiver Status Register

Adresse	Art	Funktion
$FFFA2D	R/W	Transmitter Status Register
$FFFA2F	R/W	USART Data Register
	ACIA 6850 (Tastatur)	
$FFFC00	R/W	Control Register
$FFFC02	R/W	Data Register
	ACIA 6850 (MIDI-Schnittstelle)	
$FFFC04	R/W	Control Register
$FFFC06	R/W	Data Register

Für die zum Buch gehörende Interfacekarte liegt auf der Basisadresse FA 0000 = 16384000

5.3 Direct Memory Access – DMA

Um größere Datenmengen von einem Peripheriegerät in den Hauptspeicher zu übertragen, kann man ein Programm schreiben, mit dem die Zentraleinheit diese Aufgabe ausführt. Wie bei jedem anderen Programm auch, muß der Prozessor zunächst die Programmbefehle aus dem Hauptspeicher holen, die Daten von einem Port einlesen und anschließend in den Hauptspeicher schreiben. Zusätzlich muß noch die Programmschleife verwaltet werden. Die Ausführung eines solchen Programms ist sehr zeitaufwendig. Daneben gibt es ein Verfahren, bei dem ein Zusatzbaustein die Daten von dem Peripheriegerät liest und direkt in den Hauptspeicher ablegt. Da dieser Zusatzbaustein keine anderen Aufgaben hat, ist hierfür kein Programm erforderlich und die Funktion wird sehr schnell ausgeführt. Generell sind Funktionseinheiten, die nur für spezielle Aufgaben vorgesehen sind, und bei denen die Funktion durch Hardware realisiert wird, viel schneller als eine entsprechende softwaregesteuerte Baugruppe.

Der Zusatzbaustein zum Transferieren von Daten in den Hauptspeicher ist der sogenannte DMA-Controller (Direct Memory Access – Direkter Speicherzugriff). Während einer DMA-Operation verfügt der DMA-Controller über den Systembus. Wenn die Zentraleinheit eine DMA-Anforderung erhält, teilt sie dem DMA-Controller nur noch die Anzahl der zu übertragenden Bytes und die Startadresse im Speicher mit, wo die Daten hingeschrieben werden sollen. Der DMA-Controller übernimmt dann den Datentransfer, während die Zentraleinheit in dieser Zeit angehalten wird.

5.4 Die Interruptstruktur

Ein Computer arbeitet immer ein starres Programm ab, das normalerweise nicht unterbrochen werden kann. Für viele Anwendungen ist es aber wünschenswert, das gerade ablaufende Programm zu unterbrechen und eine andere Aufgabe zu erledigen. Als Beispiel sei hier Uhreninterrupt angeführt: Durch den Impuls des Zählers wird 200 mal in der Sekunde eine Unterbrechung ausgelöst, mit deren Hilfe beim Atari ST z.B. die Systemuhr realisiert wird.

Die Interruptstruktur des 68000 wurde in Kapitel 4.2 bereits erläutert. Hier seien noch einmal die wichtigsten Informationen zusammengefaßt: Der Prozessor hat drei Interruptanschlüsse IPL0 bis IPL2. Damit werden sieben Interruptebenen festgelegt. Sind alle Anschlüsse gleich Null, so liegt kein Interrupt an, der Prozessor arbeitet im Normalbetrieb. Geht eine der Leitungen auf High, so leitet dies eine Interruptverarbeitung ein. Ein Interrupt kann nur durch einen Interrupt höherer Priorität unterbrochen werden.

Die Interrupts werden in sogenannte Auto- und Non-Autovektoren unterteilt. Ein Autovektor liegt vor, wenn bei einer Interruptanforderung durch die Leitungen IPL0 - IPL2 der VPA-Anschluß (Valid Peripheral Adress, gültige Peripherieadresse) Low ist. Hierbei ist die Interrupt-Nummer gleich der Priorität. Bei den Non-Autovektoren geht der DTACK-Anschluß (Data Transfer Acknowledge, Datentransfer erkannt) auf Low. Die Interrupt-Nummer muß der CPU in diesem Fall von der Interrupt-Quelle auf dem Datenbus mitgeteilt werden.

Zur Verarbeitung der Interrupts liegt am Anfang des Speichers eine Tabelle mit den sogenannten Interrupt-Vektoren. Von den drei Interruptanschlüssen des 68000 werden beim Atari ST aber nur zwei verwendet: IPL1 und IPL2. Damit sind nur drei Interruptebenen (2, 4 und 6) möglich, vor allem kann kein NMI (nicht maskierbarer Interrupt) ausgelöst werden. Die einzelnen Interruptebenen sind durch folgende Funktionen belegt:

Ebene 2: HBLANK Interrupt (Horizontal Blanking), Autovektor
Ebene 4: VBLANK Interrupt (Vertikal Blanking), Autovektor
Ebene 6: Interrupts des MFP 68901, Non-Autovektor

Die Interrupt-Vektoren beim Atari ST

Vek.-Nr.	Adresse	Funktion
0	$0000	Reset, Wert des Supervisor-Stackpointers
1	$0004	Reset, Wert des Programm-Zählers
2	$0008	Busfehler
3	$000C	Adreßfehler
4	$0010	Undefinierter Befehl
5	$0014	Division durch Null
6	$0018	CHK-Interrupt
7	$001C	Befehl TRAPV

Vek.-Nr.	Adresse	Funktion
8	$0020	Privilegierter Befehl im User-Modus
9	$0024	Trace
10	$0028	Line A Emulator
11	$002C	Line F Emulator
12	$0030	reserviert
13	$0034	reserviert
14	$0038	reserviert
15	$003C	Nicht initialisierter Non-Autovektor
16	$0040	
bis		für künftige 680xx-Versionen reserviert
23	$005C	
24	$0060	falscher Interrupt
25	$0064	Autovektor 1, nicht benutzt
26	$0068	Autovektor 2, HBLANK Interrupt
27	$006C	Autovektor 3, nicht benutzt
28	$0070	Autovektor 4, VBLANK Interrupt
29	$0074	Autovektor 5, nicht benutzt
30	$0078	Autovektor 6, nicht benutzt
31	$007C	Autovektor 7, (NMI) nicht benutzt
32	$0080	TRAP #0
33	$0084	TRAP #1 (GEMDOS)
34	$0088	TRAP #2 (GEM)
35	$008C	TRAP #3
36	$0090	TRAP #4, illegaler Befehl
37	$0094	TRAP #5
38	$0095	TRAP #6
39	$009C	TRAP #7
40	$00A0	TRAP #8
41	$00A4	TRAP #9
42	$00A8	TRAP #10
43	$00AC	TRAP #11
44	$00B0	TRAP #12
45	$00B4	TRAP #13 (BIOS)
46	$00B8	TRAP #14 (XBIOS)
47	$00BC	TRAP #15
64	$0100	MFP: Busy, Paral. Schnittst., I/O 0, gesperrt
65	$0104	MFP: DCD, ser. Schnittst., I/O 1, gesperrt
66	$0108	MFP: CTS, ser. Schnittst., I/O 2, gesperrt
67	$010C	MFP: Signal vom Blitter, I/O 3, gesperrt

Vek.-Nr.	Adresse	Funktion
68	$0110	MFP: Bautr., ser. Schnittst., Timer D, gesperrt
69	$0114	MFP: 200 Hz Systemtakt, Timer C, aktiv
70	$0118	MFP: Tastatur/MIDI (ACIA), I/O 4, aktiv
71	$011C	MFP: FDC-Ready/ACSI-Bus, I/O 4, gesperrt
72	$0120	MFP: DE-Signal (Horiz. Sync.), Timer B, aktiv
73	$0124	MFP: Sendefehler, ser. Schn., XMIT-Er, aktiv
74	$0128	MFP: Sendepuffer leer, ", XMIT-PE, aktiv
75	$012C	MFP: Empfangsfehler, ", RCV-Er, aktiv
76	$0130	MFP: Empfangspuf. voll, ", RCV-PE, aktiv
77	$0134	MFP: unbenutzt, gesperrt
78	$0138	MFP: RI Signa., ", I/O 6, gesperrt
79	$013C	MFP: Monochrom Monitor Detect, I/O 7, gesperrt

Alle mit MFP bezeichneten Interrupt sind für den MFP 68901 reserviert, der in Kapitel 4.3 genauer beschrieben wurde. Von den 16 möglichen Interrupts des Bausteins werden aber nur einige vom Betriebssystem des Atari ST unterstützt (in der obigen Tabelle mit aktiv bezeichneten). Der Zeitinterrupt wird 200mal in der Sekunde ausgelöst und inkrementiert bei jedem Aufruf eine Speicheradresse, womit die Systemuhr des Atari ST aufgebaut wird. Im Betriebssystem wird dieser Interrupt aber auch für die Auto-Repeat-Funktion der Tastatur verwendet. Der Autovektor HBLANK (Horizontal Blanking) Interrupt wird jedesmal ausgelöst, wenn der Elektronenstrahl im Monitor eine Zeile geschrieben hat und der Strahl beim Rücklauf zum Anfang der Zeile dunkel getastet (ausgeschaltet) wird. Dies geschieht bei Monochrom- alle 28 µs, bei Farbmonitoren alle 64 µs. Dieser Interrupt wird im Normalbetrieb beim Atari ST wegmaskiert (abgeschaltet). Der Autovektor VBLANK (Vertikal Blanking) Interrupt wird immer dann aufgerufen, wenn der Elektronenstrahl im Monitor ein Bild vollständig geschrieben hat und der Strahl beim Rücklauf zur oberen linken Ecke des Monitors dunkel getastet wird. Beim Monochrom- alle 14 ms, bei Farbmonitoren alle 20 ms. Der VBLANK- und der Systemtakt sind also die beiden einzigen Interrupts, die im Atari ST regelmäßig aufgerufen werden.

5.5 Weitere Schnittstellen

Der Atari ST ist von seinem Konzept her ein geschlossenes System, in dem alle benötigten Schnittstellen bereits vorhanden sind. Bis auf die eingeschränkten Möglichkeiten des Cartridge-Systems wurde auf einen echten Erweiterungsbus verzichtet, wie ihn etwa der legendäre Apple IIe oder die PC-Familie aufweisen. Im Mega-Atari ist man von diesem Konzept aber auch wieder abgewichen, denn auf der Hauptplatine dieses Computers ist der gesamte Systembus auf eine Steckerleiste herausgeführt. Es erweist sich eben immer als Vor-

teilhaft, einen Computer erweitern zu können. Die meisten Anwender des Atari ST werden einen solchen Erweiterungsbus aber gar nicht vermissen, denn die Anzahl der vorhandenen Schnittstellen ist beeindruckend. Im folgenden sollen die Schnittstellen des Atari ST in einem kurzen Überblick dargestellt werden.

Tastatur

Die Tastatur ist mit einem eigenen Prozessor ausgerüstet und arbeit völlig unabhängig vom Rest des Atari ST, über den sie mit einer seriellen Schnittstelle verbunden ist. Man nennt dieses System daher auch IKBD (Intelligent Keyboard Device, intelligentes Tastaturgerät). Der im Tastatursystem verwendete 6301V1 ist ein Single-Chip-Prozessor aus der 6800-Familie und enthält neben einer 8-Bit CPU 4 Kbyte ROM, 128 Byte RAM, einen 16-Bit Timer, eine serielle Schnittstelle und 4 I/O-Ports mit 29 Leitungen.

Die Port-Leitungen werden unter anderem für die Tastaturmatrix und den Anschluß der Maus und des Joysticks verwendet. Mit dem Timer des Tastaturprozessors wird die Systemuhr realisiert. Am Atari ST wird der Datentransfer mit dem IKBD über eine der beiden ACIA-Bausteine abgewickelt. Bei der Übertragung von Informationen des IKBD an den 68000 wird zwischen normalen Tastaturcodes und allen anderen Daten (Maus-, Joystickpositionen und Systemzeit) unterschieden. Die Tastaturcodes werden ohne weitere Informationen gesendet, alle anderen Daten beginnen mit einem Kennbyte im Bereich von $F6 bis $FF.

Der Hauptprozessor kann auch Befehle an das IKBD abschicken, die z.B. das System reseten, die Mausempfindlickkeit ändern, die Systemzeit setzen oder abfragen, Statusinformationen anfordern oder ähnliches bewirken.

Das MIDI-Interface

Das Music Instrument Digital Interface (MIDI, digitale Schnittstelle für Musikinstrumente) ist eine spezielle serielle Schnittstelle, die mit dem zweiten ACIA-Baustein aufgebaut ist. Die MIDI-Norm wurde speziell für den Anschluß digitaler Musikinstrumente eingeführt.

Im allgemeinen sind Musikinstrumente (oder sollte man besser Musikcomputer sagen?), die eine digitale Klangerzeugung haben oder digital steuerbar sind, mit einer solchen Schnittstelle ausgerüstet. Über einen Computer mit derselben Schnittstelle lassen sich diese Instrumente damit fernbedienen bzw. programmieren.

Parallele Schnittstelle

Die parallele Schnittstelle des Atari ST entspricht im wesentlichen der Centronics-Norm zum Anschluß eines Druckers (eine vollständige Beschreibung dieser Norm findet sich in Kapitel 3.6). Es werden aber nicht alle Kontrolleitungen der Centronics-Norm unterstützt, sondern neben den Datenleitungen nur noch das Strobe- und das Busy-Signal. Die Daten- und Strobe-Leitungen werden vom PSG- (Programmable Sound Generator, siehe Kapitel 4.6), die Busy-

Leitung vom MFP-Chip (Multifunction Peripheral, siehe Kapitel 4.3) zur Verfügung gestellt (Bild 5.4). Alle Signale haben TTL-Pegel.

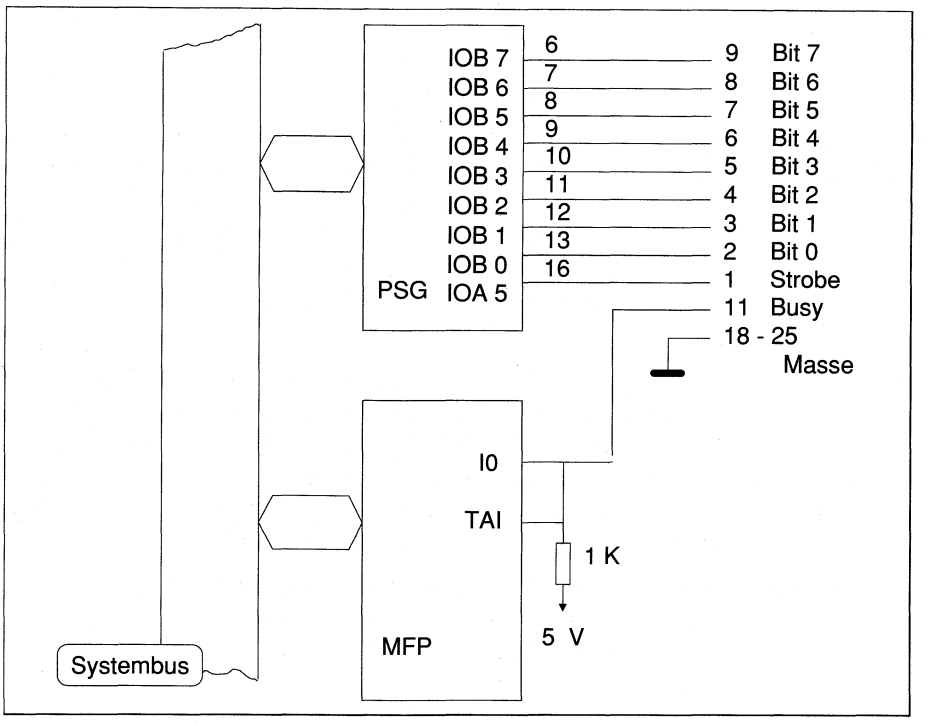


Bild 5.4: Blockschaltbild der Parallel-Schnittstelle

Da mit dem Port B des PSG-Bausteins sowohl Ein- als auch für die Ausgabe von Daten möglich ist, kann man mit der Parallelschnittstelle des Atari ST auch Daten empfangen. Das Betriebssystem unterstützt diese Betriebsart sogar mit entsprechenden BIOS-Funktionen. Die Busy-Leitung der Schnittstelle ist außer mit der Leitung I0 auch noch mit dem TAI-Eingang (Timer A Input) des MFP verbunden. Dieser Timer wird vom Betriebssystem des Atari ST nicht verwendet und es ist daher möglich, diese Leitung zur Ereignis- oder Pulsweitenmessung zu verwenden.

Serielle Schnittstelle

Die serielle Schnittstelle des Atari ST entspricht in etwa der RS-232-Norm, und arbeitet dabei als Datenendgerät. Das bedeutet, die Daten werden über die TDX-Leitung ausgegeben und über die RDX-Leitung empfangen (eine ausführliche Beschreibung der seriellen Verbindung findet sich in Kapitel 3.7). Daneben stehen fünf Handshake-Signale zur Verfügung.

Name	E/A	Funktion
RTS	A	Request To Send, Sendeteil einschalten. Bei Low wird die Datenübertragung angehalten.
CTS	E	Clear To Send, Sendebereitschaft. Daten können gesendet werden, die Gegenstation ist bereit.
DCD	E	Data Carrier Detect, Empfangssignale liegen vor Die Gegenstation empfängt Daten.
DTR	A	Data Terminal Ready, Datenendgerät betriebsbereit. Der Gegenstation wird mit High die Betriebsbereitschaft gemeldet.
RI	E	Ring Indikator, ankommender Ruf. Die Gegenstation meldet will einen Datentransfer einleiten.

E/A zeigt an, ob es sich vom Atari ST aus gesehen um eine Eingabe- (E) oder eine Ausgabeleitung (A) handelt.

Wie bei der parallelen, so teilen sich auch bei der seriellen Schnittstelle der MFP und der PSG die Aufgaben, wie in Bild 5.5 gezeigt.

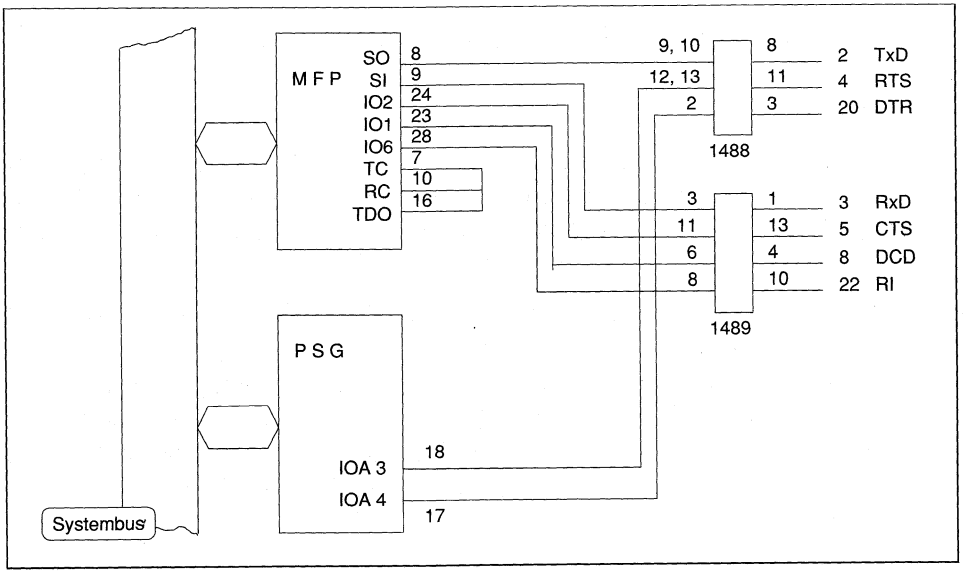


Bild 5.5: Blockschaltbild der seriellen Schnittstelle

DMA – Direkt Memory Access

Wie der 68000 den DMA-Betrieb unterstützt wurde in Kapitel 4.2 erläutert. Der DMA-Controller im Atari ST gehört zu den Custom-Chips und ist am Anfang dieses Kapitels erläutert. Der herausgeführte DMA-Port ist primär für eine Festplatte vorgesehen, aber natürlich könnten auch andere Geräte angeschlossen werden. Auf der Hauptplatine ist der DMA-Baustein noch mit dem FDC-Chip verbunden, wie in Bild 5.6 gezeigt.

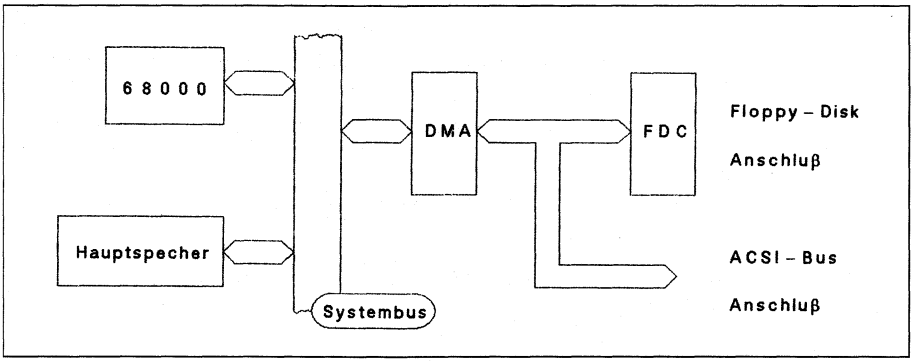


Bild 5.6: Blockschaltbild der am DMA-Betrieb beteiligten Baugruppen

ROM-Erweiterung

Eine für dieses Buch sehr wichtige Schnittstelle ist das Catridge-System für ROMs. Mit den vorhandenen Leitungen steht Speicherplatz für 128 Kbyte (von $FA0000 bis $FBFFFF) zur Verfügung. Bild 3.4 in Kapitel 3 zeigt die Anschlüsse. Es sind zwei Selektleitungen vorhanden, die den Speicherbereich in zwei 64 Kbyte Blöcke aufteilen.

Die Hardware des Atari ST ahndet jeden Schreibversuch in diesem Speicherbereich mit einem Busfehler.

Die diesem Buch beiliegende Interfacekarte wird in der ROM-Erweiterung betrieben, w obei alle Schreiboperationen auf die Karte über einen Teil des Speichers ausgeführt werden müssen. Hierbei sind 256 Speicherplätze (8 Bit des Adreßbusses) als Übergabekanäle für Daten reserviert.

6 Verarbeitung analoger Daten

6.1 Analoge Signale und analoge Pegel

Mit seinen Sinnen kann der Mensch von den Wechselwirkungen seiner Umgebung nur einen geringen Teil erfassen. Für Größen wie elektrische Spannungen, elektrische Ströme, radioaktive Strahlungen oder elektromagnetische Wellen fehlen ihm jegliche Sinnesorgane. Selbst physikalische Größen, die eine Reaktion auf die Sinnesorgane auslösen, werden entweder nur in speziellen Bereichen oder bei begrenzten Amplituden erfaßt. So kann das menschliche Auge zwar von dem Licht der Sterne bis zum grellen Sonnenlicht alles wahrnehmen, aber ihm bleiben Lichtreize im infraroten oder ultravioletten Bereich verborgen. Zahlreiche Signale können von Lebewesen erst bei genügend großen Amplituden erfaßt werden. Beispiele hierfür sind die Größen: Druck, Schallpegel oder das chemische Verhalten einiger Substanzen. Mit dem Fortschritt der Technik wurden für alle physikalischen Meßgrößen Sensoren entwickelt. Seit der Entstehung der Rechner können diese Umweltsignale für spezielle Anwendungen ausgewertet werden. Beispiele hierfür sind neben einfachen Regelvorgängen wie die Steuerung einer Zentralheizung auch komplexe Verfahren wie ein Antiblokkiersysteme im Auto oder eine satellitengestützte Bilddatenverarbeitung zum Erkennen von Trockengebieten auf der Erde.

Die Rechnerverarbeitung analoger Signale ist in allen Anwendungsbereichen nach demselben Schema aufgebaut: Ein Sensor erfaßt die analoge Meßgröße (z.B. Temperatur, Lichtstärke oder Druck) und wandelt diese in ein genormtes Signal (z.B. in eine Spannung mit dem Pegel von 0 bis 10 Volt) um. Das genormte Signal gelangt vom Sensor zu einem Analog-/Digital-Wandler (ADC: Analog to Digital Converter), der die analoge Größe in einen digitalen Wert umwandelt. Der Rechner führt dann die gewünschte Verarbeitung durch und zeigt die ermittelte Ausgangsgröße an.

Gegebenenfalls wandelt ein Digital-Analog-Wandler (DAC: Digital to Analog Converter) die Ausgangsgröße wieder in einen Analogwert um, damit ein geschlossener analoger Regelkreis entsteht.

Der folgende Abschnitt beschäftigt sich mit dem Messen, Umwandeln und Darstellen analoger Meßgrößen.

6.2 Der Operationsverstärker

Eines der wichtigsten Elemente zum Aufbereiten und Verarbeiten analoger elektrischer Größen ist der Operationsverstärker. Unter diesem Element versteht man einen integrierten Schaltkreis, der aus Transistoren und Widerständen und eventuell Kondensatoren aufgebaut ist. Operationsverstärker zeigen bei der analogen Verarbeitung ein nahezu ideales Verhalten, da die auf dem Chip integrierten Transistoren absolut gleichartige elektrische Eigenschaften haben. Die enge Anordnung auf den Siliziumchip, der gleichartige Fertigungsprozeß und die lokale Spannungsversorgung lassen ungewollte Störeinflüsse fast vollkommen verschwinden. Temperaturschwankungen, wechselnde Ströme und unterschiedliche Spannungswerte wirken sich stets gleich auf alle im Chip befindlichen Halbleiter aus und kompensieren sich gegenseitig. Obwohl Operationsverstärker teilweise sehr einfach aufgebaut sind, lassen sie sich wegen der vorher genannten Randbedingungen kaum diskret, also durch einzelne Widerstände und Transistoren, realisieren.

Jeder Operationsverstärker besteht im Eingang aus einem Differenzverstärker. Jeweils zwei Eingänge des Operationsverstärkers werden auf die Basen von Transistoren oder die Gates von Feldeffekt-Transistoren geführt (Bild 6.1a). Die Emitter der Eingangstransistoren sind miteinander verbunden und führen über einen Widerstand zur negativen Versorgungsspannung. Durch diesen speziellen Schaltungsaufbau verteilt sich der über dem Widerstand R3 fließende Strom über die beiden Transistoren. Dabei hängt der Spannungsabfall an den Kollektorwiderständen R1 und R2 von der Differenz der Eingangsspannung zwischen E1 und E2 ab. Bei integrierten Operationsverstärkern haben die beiden Eingangstransistoren über weite Spannungs- und Temperaturbereiche stets die gleiche Basis-Emitter-Spannung (Ube). Dadurch genügen in der Regel bereits kleine Differenzspannungen am Eingang, um den Stromfluß durch die Kollektorwiderstände anzusteuern.

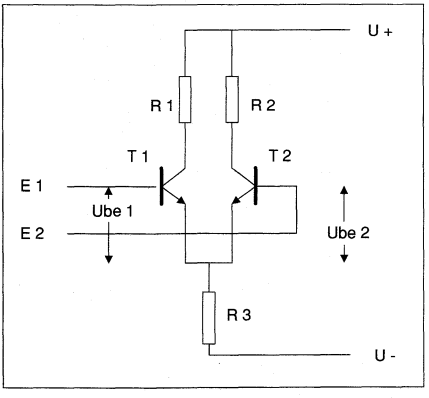


Bild 6.1a: Prinzipschaltung eines Differenzverstärkers

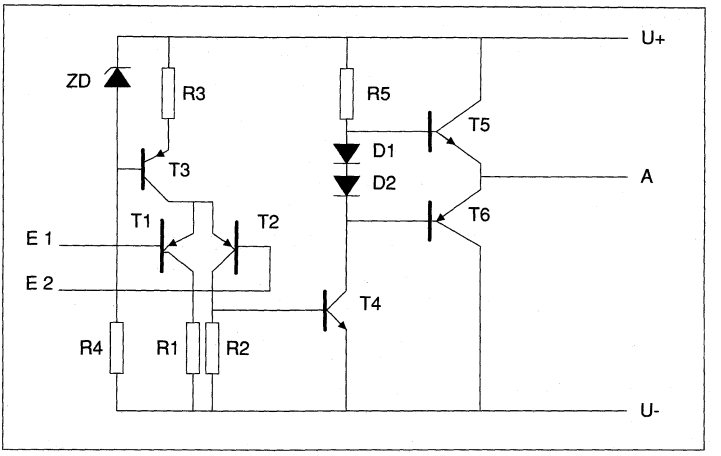


Bild 6.1b: Ein Operationsverstärker besteht aus einem Differenzverstärker und einem nachgeschalteten Gegentaktverstärker

Bild 6.1b zeigt den realistischen Aufbau eines Operationsverstärkers in einer einfachen Form. Der gemeinsame Emitterwiderstand wurde dabei durch eine Stromquelle ersetzt. Über die Zenerdiode (ZD) wird die Basis des Transistors T3 auf einem konstanten Potential zur positiven Versorgung gehalten. Damit fließt aus seinem Kollektor ein konstanter Strom, der durch den Widerstand R3 vorgegeben wird. Dieser Strom verteilt sich auf die Transistoren T1 und T2. Eine Differenzspannung am Eingang des Operationsverstärkers erzeugt einen Spannungsabfall am Widerstand R2. Die folgenden Transistoren T4, T5 und T6 verstärken die auftretende Spannung und führen sie dem Ausgang zu. Eine Verringerung der Eingangsspannung E2 bei festgehaltener Spannung von E1 bewirkt auch ein Absenken der Ausgangsspannung. Man bezeichnet den Eingang E2 deshalb auch als nichtinvertierenden und E1 als den invertierenden Eingang. Operationsverstärker lassen sich als Blockschaltbild vollständig durch die beiden Eingänge, den Ausgang und die beiden Versorgungsleitungen beschreiben (Bild 6.2). Da die Ausgangsspannung nur von der Differenzspannung der Eingänge und nicht mehr von der Differenz der Eingangsspannung zu der Versorgung abhängt, stellt man Operationsverstärker häufig auch ohne die Versorgungsleitungen dar.

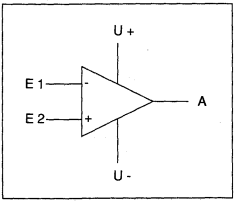


Bild 6.2: Schaltbild des Operationsverstärkers

Integrierte Operationsverstärker zeigen beim Einsatz nahezu ideale Eigenschaften, die sie für die analoge Verarbeitung besonders tauglich machen. Die folgende Tabelle verdeutlicht einige Eigenschaften in der Gegenüberstellung der idealen und realen Daten derartiger Elemente:

idealer Wert	**realer Wert**
Verstärkung unendlich	40000 bis 10000000fach
Eingangsstrom 0	10 µA – 1 pA
Eingangswiderstand unendlich	1 MOhm – 10 TOhm
Ausgangswiderstand 0	0,1 Ohm – 0,001 Ohm
Eingangsdifferenzsp. 0	2 mV – 2 µV

Die angegebenen realen Werte beziehen sich auf unterschiedliche marktübliche Typen. Die Unterschiede ergeben sich insbesondere durch das Verhalten in Hochfrequenzschaltungen oder beim Einsatz rauscharmer Verstärkerstufen. Ferner unterscheiden sich Operationsverstärker auch in ihrer Stabilität (Drift) gegenüber Temperatur- oder Spannungsänderung.

6.3 Einfache Anwendungen von Operationsverstärkern

Die im vorigen Abschnitt vorgestellten Eigenschaften eines Operationsverstärkers machen ihn zum idealen Element, wenn Analogwerte zur Rechnerverarbeitung bereitgestellt werden sollen. Die einfachste Anwendung eines Operationsverstärkers besteht im Aufbau eines Komparators. Komparatoren sind Elemente, die zwei Spannungen miteinander vergleichen und entscheiden, welche der beiden Spannungen größer ist. In einer Komparatorschaltung werden die beiden Eingänge des Operationsverstärkers mit den zu vergleichenden Spannungen verbunden.

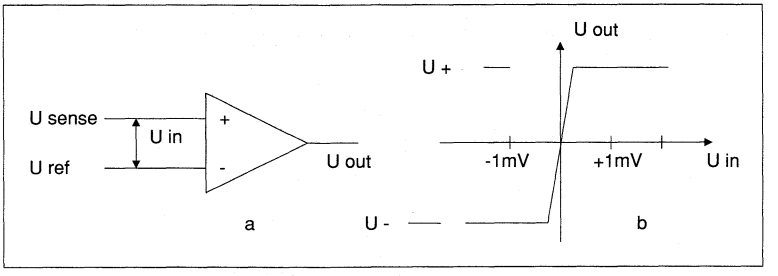


Bild 6.3: Ohne eine zusätzliche Beschaltung kann ein OPA-Komparator arbeiten

Durch die hohe Verstärkung genügt bereits eine kleine Differenzspannung an den Eingängen, um den Ausgang vollständig anzusteuern. Je nach Polarität der Eingangsdifferenz zeigt der Ausgang entweder eine positive oder negative Versorgungsspannung an (Bild 6.3).

Da bereits kleinste Eingangsspannungen zur vollständigen Aussteuerung des Ausgangs führen, neigt der im Bild 6.3 vorgestellte Komparator zu Eigenschwingungen, da die Eingangsspannungspegel selten absolut rein sind, sondern häufig ein leichtes Brummen oder Rauschen tragen. Die Eigenschwingungen treten nicht auf, wenn der Komparator eine Hysterese erhält, die größer als die eingekoppelte Brummspannung ist. Bild 6.4 zeigt einen Komparatoraufbau mit einer Hysterese. Der Rückkopplungswiderstand R2 führt einen Teil der Ausgangsspannung an den nichtinvertierenden Eingang zurück, was die Schaltschwelle künstlich erhöht. Da der Widerstand R2 die Ausgangsfunktion am Eingang unterstützt, spricht man häufig auch von einer Mitkopplung. Die Hysteresespannung hängt von dem Teilerverhältnis der Widerstände R1 und R2 sowie der Ausgangsspannung des Komparators ab.

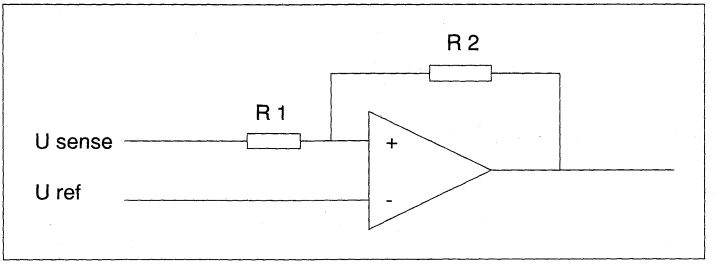


Bild 6.4: Schaltung eines Komparators mit Hysterese

Am Ausgang des Operationsverstärkers erhält man nur dann zwischen den Versorgungsspannungen liegende Werte, wenn die Differenzeingangsspannung extrem klein ist. Möchte man deshalb mit einem Operationsverstärker normale Verstärkerschaltungen bauen, so müssen die Eingänge auf eine Differenzspannung von 0 Volt abgeglichen sein.

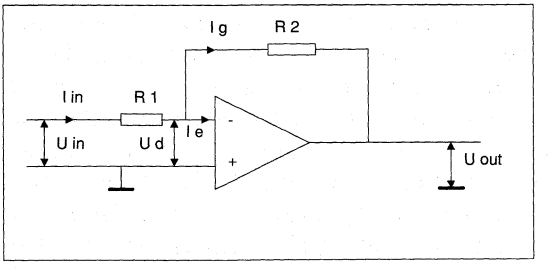


Bild 6.5: Schaltbild des invertierenden Verstärkers

Das Bild 6.5 zeigt den Aufbau eines invertierenden Verstärkers, der mit einem Operationsverstärker und zwei Widerständen aufgebaut ist.

Eine stabile Ausgangsspannung zwischen den Betriebsspannungsgrenzen wird nur dann erreicht, wenn die Eingangsdifferenzspannung Ud 0 Volt beträgt. Ferner darf in den Operationsverstärker kein Eingangsstrom hineinfließen, da sein Eingangswiderstand unendlich hoch ist (Ie = 0). Der nicht-invertierende Eingang ist auf Masse bezogen. Nach dem Ohmschen Gesetz erzeugt die Eingangsspannung Uin, je nach Wahl des Widerstands R1, einen Eingangsstrom I1. Da in den Operationsverstärker kein Strom hineinfließen kann (Ie = 0), muß der Eingangsstrom über den Widerstand R2 wieder abgeleitet werden. Die Ströme I1 (Iin) und I2 (Ig) sind also gleich. Der Strom I2 resultiert aus einer negativen Ausgangsspannung über dem Widerstand R2. Der Operationsverstärker ändert damit seinen Ausgang solange, bis sein Eingang abgeglichen ist. Aus diesen Überlegungen läßt sich die Verstärkung der Schaltung berechnen (diese und alle folgenden Gleichungen sind zum Verständnis nicht erforderlich, sie sind nur zur Vollständigkeit aufgeführt):

```
 I1    =    I2              Gleichung  a
 Uin   =    I1 * R1         Gleichung  b
-Uout  =    I2 * R2         Gleichung  c
```

Aus den Gleichungen b und c ergibt sich:

$$\frac{-Uout}{Uin} = \frac{I2 * R2}{I1 * R1}$$

Da nach a beide Ströme gleich sind, läßt sich I1 mit I2 kürzen:

$$\frac{Uout}{Uin} = \frac{R2}{R1}$$

Die Verstärkung des invertierenden Verstärkers läßt sich demnach durch die Widerstände R1 und R2 festlegen. Durch den Widerstand R2 wird die Gesamtverstärkung reduziert; es liegt eine Gegenkopplung vor.

In der Schaltung des invertierenden Verstärkers steckt noch eine weitere Anwendung. Wenn der Widerstand R1 entfällt, bewirkt der Eingangsstrom I1 direkt eine Änderung der Ausgangsspannung. Ohne den Widerstand R1 arbeitet die vorgestellte Schaltung als Strom-/Spannungs-Konverter. Beschaltet man einen Operationsverstärker mit zwei Widerständen in der in Bild 6.6 dargestellten Form, so erhält man eine Verstärkerschaltung, die am Ausgang nicht invertiert. Bei dieser Variante wird nur ein Teil der Ausgangsspannung an den invertierenden Eingang des Operationsverstärkers zurückgeführt. Dieser regelt die Ausgangsspannung Uout solange hoch, bis am Eingang eine Differenz von 0 Volt erreicht wird. Die Verstärkung der Schaltung berechnet sich aus dem Stromanteil, der von der Ausgangsspannung an den beiden Widerständen R2 und R1 erzeugt wird:

$$U_{out} = I_r * (R_2 + R_1)$$ Gleichung a

Für die Teilspannung gilt:

$$U_2 = I_r * R_2$$ und

$$U_1 = I_r * R_1$$ Gleichungen b

Da im abgeglichenen Zustand die Differenzspannung 0 Volt beträgt, ergibt sich:

$$\frac{U_1}{U_{in}} = \frac{U_{in} \quad \text{und}}{U_{out} \quad I_r * (R_2 + R_1)} = \frac{I_r * (R_2 + R_1)}{I_r * R_1}$$ Gleichung d

Diese Gleichung läßt sich auch in der folgenden Form Schreiben:

$$\frac{U_{out}}{U_{in}} = 1 + \frac{R_2}{R_1}$$ Gleichung e

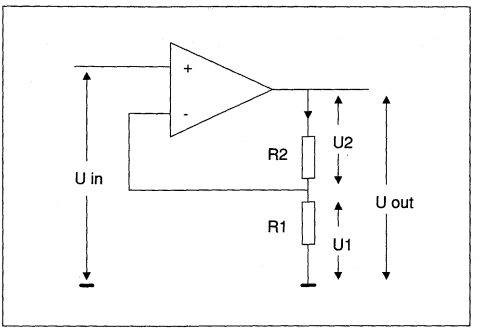


Bild 6.6: Schaltbild des nichtinvertierenden Verstärkers

Auch der nichtinvertierende Verstärker läßt sich durch die Beschaltung zweier Widerstände am Operationsverstärker aufbauen. Es gibt jedoch zwei wesentliche Unterschiede:

1. Beim invertierenden Verstärker lassen sich durch die Wahl der Widerstände auch Verstärkungen erreichen, die kleiner als 1 sind. Die Schaltung arbeitet in diesen Fällen als Abschwächer. Dagegen ist die Verstärkung des nichtinvertierenden Verstärkers ungefähr gleich 1 (z.B., wenn R1 entfällt).
2. Der nichtinvertierende Verstärker hat einen unendlich hohen Eingangswiderstand. Beim invertierenden Verstärker hängt der Eingangswiderstand von der Wahl von R1 ab.

Durch geeignete Wahl der Bauteile beim nichtinvertierenden Verstärker kann auch eine spezielle Schaltung aufgebaut werden, die als Spannungsfolger oder Elektrometer bekannt ist.

Dabei ist die Verstärkung auf den Wert 1 festgelegt. Die beiden Widerstände können dann entfallen. Der Ausgang führt in diesem Falle direkt an den invertierenden Eingang zurück. Ein derartiger Spannungsfolger wird immer dann eingesetzt, wenn eine Spannung sehr hochohmig erfaßt und einem niederohmigen Verbraucher zur Verfügung gestellt werden soll.

Die besonderen Eigenschaften des Operationsverstärkers erlauben auch den Aufbau von Schaltungen, die diskret nur mit enormen Aufwand zu realisieren wären. Zu diesen Schaltungen gehören insbesondere Rechenelemente, die Analogwerte über die vier Grundrechenarten verknüpfen. Addierer, Subtrahierer oder gar Multiplizierer und Dividierer lassen sich mit Operationsverstärkern mit wenigen Komponenten in einer Schaltung zusammensetzen. Diese Schaltungen sind jedoch wegen der fortschreitenden Rechnertechnologie heute nicht mehr so wichtig. Es ist in vielen Fällen leichter, eine Addition mittels eines Mikrocomputers durchzuführen, als auf die Addierschaltung eines Analogrechners zurückzugreifen. Darüber hinaus machen sich bei allen Analogrechnern Ungenauigkeiten und Drifterscheinungen bemerkbar. Bei der digitalen Datenverarbeitung ist jegliche Drift unerheblich. Bei den Rechnerschaltungen gibt es jedoch einige spezielle Anwendungen, die mit Hilfe eines Mikrorechners nur sehr schwer aufzubauen sind. Als Beispiel sei hier der Integrator dargestellt, der das Integral einer Eingangsspannung bildet. Vergleichbare Anwendungen sind in einem Mikrorechner bei verschiedenen Eingangsfunktionen häufig nur in numerischer Näherung lösbar.

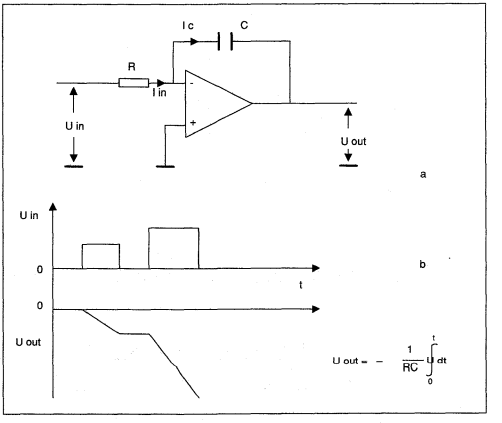


Bild 6.7a/b: Schaltbild und Signalverläufe beim Integrator

Bild 6.7a zeigt den Aufbau eines invertierenden Integrators. Die gesamte Schaltung besteht nur aus einem Operationsverstärker, der mit einem Widerstand und einem Kondensator beschaltet ist. Die Eingangsspannung Uin erzeugt über den Eingangswiderstand R einen Eingangsstrom Iin. Da dieser Strom nach der Grundbedingung aller Operationsverstärker nicht in den Eingang fließen kann, muß er über den Kondensator C abgeleitet werden. Weil ein Kondensator jedoch für Gleichspannung absolut hochohmig ist, kann nur dann ein Stromfluß

zustande kommen, wenn der Ausgang des Operationsverstärkers seine Spannung Uout verändert. Je größer der Stromfluß ist, desto stärker muß die Ausgangsspannungsänderung erfolgen. Bild 6.7b zeigt das Diagramm einer möglichen Eingangsspannung an einem Integrator mit der resultierenden Ausgangsfunktion. Jede Eingangsspannung hat eine dauernde Änderung des Ausgangs zur Folge. Je größer die Eingangsspannung ist, desto größer ist auch die Änderung der Ausgangsspannung. Der Integrator verweilt in einer Ruhestellung, wenn der Eingang 0 Volt vorgibt. Bild 6.7b gibt auch die allgemein gültige Formel für einen Integrator wieder.

Für Liebhaber mathematischer Ableitungen soll die in Bild 6.7b angegebene Formel noch hergeleitet werden, alle anderen Leser können diesen Abschnitt getrost überspringen. Über einen Kondensator fließt nur dann Strom, wenn sich das Spannungspotential an ihm ändert:

$$I = -C * \frac{dU}{dt} \qquad \text{Gleichung a}$$

Der Strom, der über den Kondensator abfließt, wird vom Spannungsabfall der Eingangsspannung am Widerstand R erzeugt:

$$I = \frac{Uin}{R} \qquad \text{Gleichung b}$$

Aus den Gleichungen 6.3a und 6.3b ergibt sich durch Gleichsetzen:

$$\frac{Uin}{R} = -C * \frac{dU}{dt} \qquad \text{Gleichung c}$$

Formt man die Gleichung 6.3c um, so folgt:

$$dU = -\frac{Uin}{R * C} dt \qquad \text{Gleichung d}$$

Integriert man diese Gleichung im Intervall von 0 bis t und zieht die Konstanten R und C vor das Integral, ergibt sich die im Bild 6.7b dargestellte Gleichung. Aus der Schaltung des invertierenden Verstärkers und des Integrators läßt sich eine gemeinsame Schaltung aufbauen, die als Mittelwertbilder oder Anstiegsbegrenzer bekannt ist. Wie im Bild 6.8a zu sehen, erfolgt die Gegenkopplung des Operationsverstärkers durch die Kombination eines Kondensators und eines Widerstands. Die Schaltung wird immer dann eingesetzt, wenn es darauf ankommt, eine genaue Verstärkung zu erreichen, man aber kurzzeitige Schwankungen unterdrücken möchte (Rauschen oder Impulsstörungen). Bei allen schnellen Eingangsänderungen wirkt der Kondensator als niederohmiger Widerstand und reduziert die Verstärkung des Systems. Stabile oder langzeitige Eingangsgrößen beeinflussen den Ladungstransport des Kondensators nicht und der Widerstand der Gegenkopplung legt den Eingangsstrom fest.

Bild 6.8b zeigt in einem Diagramm das Ausgangsverhalten des Mittelwertbilders bei einer speziellen Eingangsspannungsform.

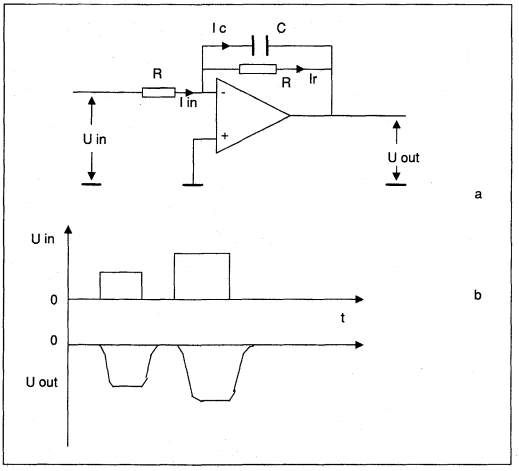


Bild 6.8a/b: Schaltbild und Signalverläufe beim Mittelwertbilder

6.4 Verfahren zur Analog-Wandlung digitaler Werte

Mikrorechner verarbeiten ihre Daten in digitaler Form, während in der Umwelt analoge Signale vorliegen. Für viele Anwendungen müssen die digitalen Informationen aus dem Computer in analoge Regel- und Stellgrößen umgewandelt werden. Die wohl verbreitetste Methode zur Analog-Wandlung besteht darin, über ein speziell geartetes Widerstandsnetzwerk digitale Ströme oder Spannungswerte in einen Analogwert zu überführen. Eines dieser Widerstandsnetzwerke, die für die Umwandlung geeignet sind, ist das R/2R-Netzwerk. Es besteht aus einer Anzahl von Widerständen, die konstante Widerstandswerte R und 2R (oder auch Serienschaltungen von zwei Widerständen je R) haben. Wenn man diese Widerstände in der im Bild 6.9 gezeigten Art und Weise zusammenbringt, erhält man eine Schaltung, die von Stufe zu Stufe eine Halbierung der Stromwerte ergibt. Jede Spannung bewirkt in der folgenden Stufe nur noch eine Halbierung, da an jeder Stelle der Einspeisung ein Spannungsteiler von 2R zu 2R vorliegt. So bewirkt das Einprägen der Referenzspannung am Schalter S-MSB (Most Significant Bit) am Ausgang des Netzwerks genau die halbe Referenzspannung. Dieses Verhalten ist leicht einzusehen, wenn man die Summenwirkung aller weiteren Widerstände berechnet und gleichzeitig annimmt, die anderen Schalter seien mit Masse verbunden. Der Wert aller restlichen Widerstände beträgt nämlich 2R: Die Referenzspannung wird so genau durch 2 geteilt. Der nächste Schalter (S) vermag den Ausgang nur noch zu einem Viertel und der letzte Schalter S-LSB (Least Significant Bit) erscheint am Ausgang nur noch mit einem Achtel als Beitrag.

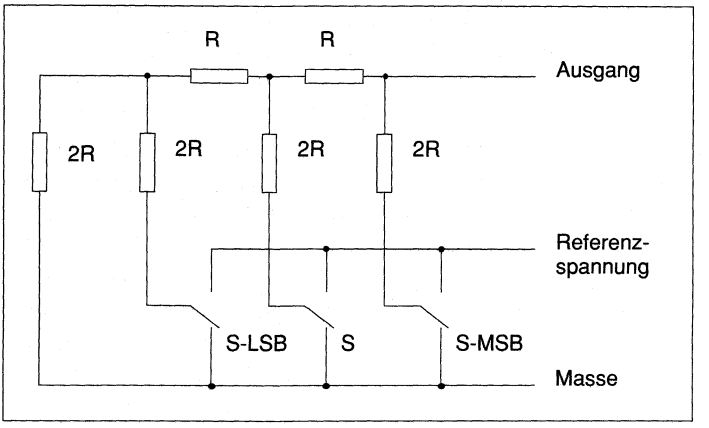


Bild 6.9: Schaltbild eines R/2R-Netzwerkes

R/2R-Netzwerke lassen sich heutzutage in integrierter Form leicht realisieren. Durch die Integration gelingt ein hinreichend guter Abgleich der Widerstände zueinander (wobei der absolute Wert nicht so wichtig ist) und die Temperaturdifferenz der Widerstände gegeneinander bleibt gering. Die Schalter lassen sich durch MOS-FETs aufbauen. Je nach Anzahl der Stufen dienen derartige Schaltungen zur Analog-Wandlung für Digitalgrößen von 6, 8, 10, 12 oder gar 14 Bit. Mit zunehmender Stufenzahl müssen auch die Widerstände immer präziser werden, und der Preis der Netzwerke nimmt rapide zu.

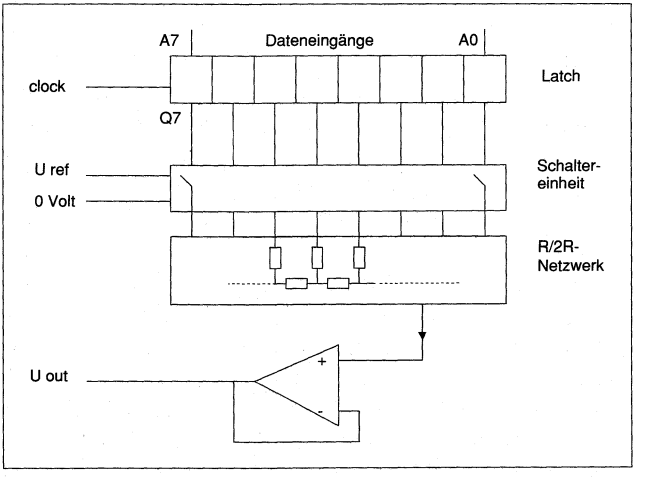


Bild 6.10: Prinzipieller Aufbau eines DAC (Digital to Analog Converter) mit einem R/2R-Netzwerk

Bild 6.10 zeigt in einem Blockschaltbild den Aufbau eines Digital-/Analog-Wandlers (DAC) für die Datenbreite von 8 Bit (1 Byte). Die digitale Information wird in einem achtfachen Zwischenspeicher (Latch) festgehalten und dort einem MOS-FET-Schaltwerk zugeführt. Je nach logischer Information der Ausgänge des Zwischenspeichers (Q0-Q7) verbinden die Schalter das Masse-Potential oder die Referenzspannung mit dem jeweiligen Ausgang. Das nachgeschaltete R/2R-Netzwerk erzeugt an seinem Ausgang die Spannungsinformation, die dem Digitalwert entspricht. Um jede Verfälschung dieses Signals zu vermeiden, übernimmt ein Operationsverstärker als Spannungsfolger diese Ausgangsgröße und stellt sie zur weiteren Verarbeitung zur Verfügung. Die Benutzung von Widerstandsnetzwerken zur Umwandlung digitaler Größen in analoge beschränkt sich keineswegs auf die R/2R-Netzwerke. Je nach gewünschter Auflösung erfüllen auch anders gestaltete Netzwerke die Aufgaben der Analogkonversion. Neben dem Einsatz von Widerstandsnetzwerken gibt es noch zahlreiche weitere Verfahren, um aus einer digitalen Information ein analogen Signals zu erzeugen. Eines der wichtigsten Verfahren ist die Frequenz-/Spannungs-Wandlung.

Jede digitale Verarbeitung erlaubt die Ausgabe von Impulsen mit konstanter Impulsdauer. Es ist eine leichte Aufgabe, aus einem Digitalwert eine Frequenz zu erzeugen, die proportional zum Digitalwert ist. So kann dem Digitalwert 100 eine Frequenz von 1 kHz und dem Digitalwert von 1000 entsprechend eine Frequenz von 10 kHz zugeordnet werden. Wenn es gelingt, die Ausgangsimpulsbreite und den Ausgangsspannungspegel konstant zu halten, entspricht der Mittelwert der Impulse über längere Zeit genau dem Analogwert der Frequenz. Für die Analogkonversion benötigt man somit nur einen Ausgangsschalter, der je nach Taktimpuls zwischen Masse und einer Referenzspannung hin- und herschaltet, und einen Mittelwertbilder, der den Analogwert erzeugt.

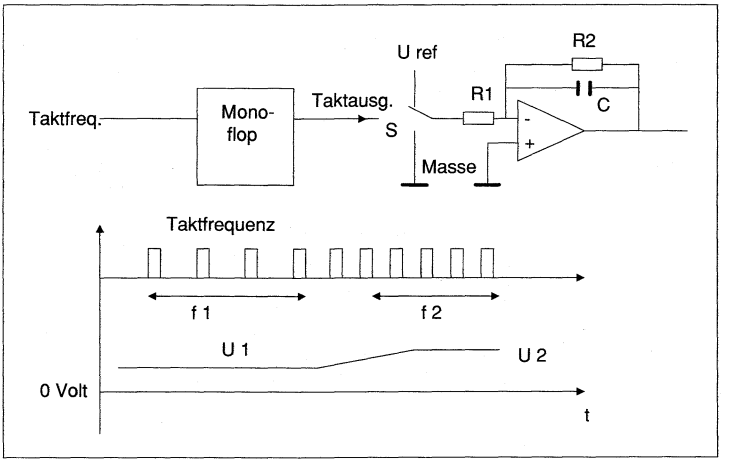


Bild 6.11: Frequenz-/Spannungs-Wandler

Bild 6.11 zeigt den Prinzipaufbau eines derartigen Analogwandlers und sein Verhalten bei zwei unterschiedlichen Frequenzen. Die Taktfrequenz gelangt über ein Monoflop an den Eingang des Mittelwertbilders. Das Monoflop erzeugt aus jeder Taktimpulsflanke der Eingangsfrequenz eine konstante Impulsdauer. Je nach Zustand des Ausgangs am Monoflop legt der nachfolgende Schalter den Eingang des Mittelwertbilders entweder an Masse oder an die Spannungsreferenz. Nach längerer Zeit entspricht der Mittelwert seines Ausgangs dann der Eingangsfrequenz. Dieses Umsetzverfahren ist unabhängig von der Stabilität der Widerstände, der richtige Mittelwert wird aber erst nach recht langer Zeit erzeugt. Das Verfahren der Frequenz-/Spannungs-Wandlung ist deshalb für schnelle Anwendungen nicht geeignet.

6.5 Analog-/Digital-Wandlung durch das Rampenverfahren

Das Wandeln analoger Größen in digitale Informationen ist noch wichtiger als die Konversion digitaler Größen. Überaus häufig werden analoge Werte im Mikrocomputer verarbeitet, was eine Analog-/Digital-Wandlung voraussetzt. Beim Rampenverfahren wird eine analoge Größe durch den direkten Vergleich zweier Analogwerte gewandelt. Einer der Analogwerte ist der zu messende Eingangswert (Ist-Wert), der zweite entspricht einem Referenzwert, den der Mikrorechner zugrundelegt (Soll-Wert). Wie Bild 6.12 zeigt, besteht ein ADC nach dem Rampenverfahren aus einem Zähler, dessen maximale Zähllänge der gewünschten Auflösung entspricht, aus einem DAC und aus einem Komparator. Der Konversionsvorgang beginnt mit dem Löschen des Zählers. Die resultierende Ausgangsspannung am DAC ist damit 0 Volt. Der nachgeschaltete Komparator vergleicht die DAC-Spannung mit der zu messenden Eingangsspannung. Sofern die Eingangsspannung Uin größer als 0 Volt ist, zeigt der Komparator am Ausgang einen positiven Wert an. Nach dem Rücksetzen des Zählers erhält der Eingang laufend Taktimpulse, die den Zähler und damit den Ausgang des DACs ansteigen lassen. Trägt man die Ausgangsspannung am DAC gegen die Zeit auf, entsteht eine linear ansteigende Rampe (Single Slope).

Der Konversionsvorgang wird genau dann beendet, wenn der Komparator von einem negativen zu einem positiven Ausgangswert umschaltet. An dieser Stelle hat die Ausgangsspannung des DAC die Eingangsspannung übertroffen. Der Digitalwert, der jetzt im Zähler steht, entspricht der Eingangsspannung, die auf diese Weise mit der Genauigkeit eines Quantisierungsschritts konvertiert wurde. Das Rampenverfahren ist eines der wichtigsten Konversionsverfahren und läßt sich in einfacher Weise realisieren. Es arbeitet dann richtig, wenn sich die Eingangsspannung während des Konversionsvorgangs nicht verändert. Die Konversionszeit hängt von der Eingangsspannung ab, was der große Nachteil dieses Verfahrens ist. Bild 6.13 zeigt das Konversionsverfahren für zwei unterschiedliche Eingangsspannungen. Mit zunehmender Eingangsspannung steigt die Konversionszeit immer mehr an.

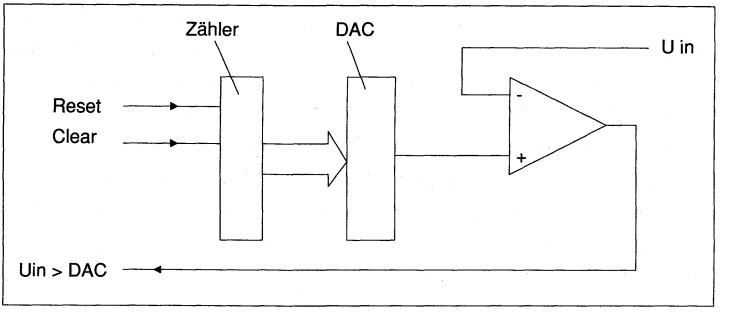


Bild 6.12: Prinzipieller Aufbau eines ADC (Analog to Digital Converter)

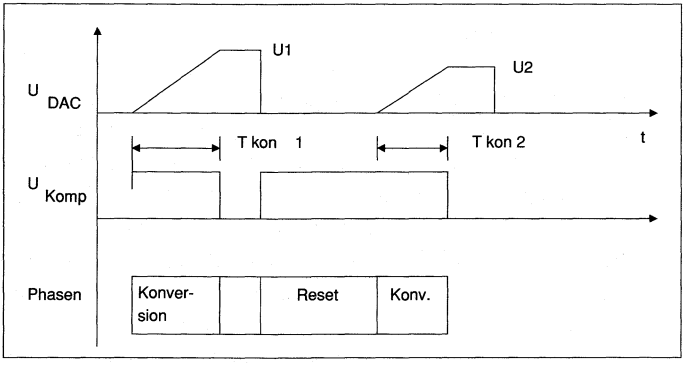


Bild 6.13: Rampenverfahren zur Analog/Digital-Wandlung.
Die Konversionszeit hängt von der Höhe der Eingangsspannung ab.

Eine häufig verwendete Variante des Rampenverfahrens mit einem DAC entsteht, wenn man den Zähler und das DAC durch einen Integrator ersetzt. Wie Bild 6.14 zeigt, wird der Integrator an eine konstante, negative Referenzspannung geschaltet, damit an seinem Ausgang eine linear ansteigende Rampe erscheint.

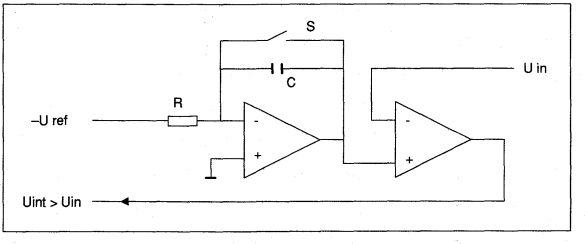


Bild 6.14: Schaltbild eines ADCs nach dem Rampenverfahren

Die Anfangsbedingung stellt man ein, indem der Kondensator C über den Schalter S entladen wird. Die Messung beginnt, wenn der Kurzschluß aufgehoben ist und der Ausgang des Komparators linear ansteigt. Genau wie im Falle des Rampenverfahrens mit dem DAC schaltet der Komparator um, wenn die Rampenspannung die Eingangsspannung übertrifft. Wie aus Bild 6.13 ersichtlich, entspricht die Anstiegszeit der Rampe von 0 Volt bis Gleichstand der Höhe der Eingangsspannung. Mißt man die Zeit bis der Komparator umschaltet, hat man die Eingangsspannung bestimmt. Das Verfahren konvertiert die Eingangsspannung in eine Zeitdauer.

6.6 Doppelrampenverfahren zur Analog-/Digital-Wandlung

Das im vorigen Abschnitt dargestellte Konversionsverfahren mit einem Integrator beinhaltet einige Nachteile, die zu Fehlmessungen beitragen können. So hängt der Anstiegswinkel der Ausgangsrampe des Integrators (siehe Formel des Integrators in Abschnitt 6.2) von der Referenzspannung und von den Werten des Widerstands und des Kondensators ab. Diese drei Größen gehen direkt in die Rampendauer ein und bestimmen damit den Umsetzwert. Die Größen des Widerstands und des Kondensators können über kurze Zeit gut konstant gehalten werden, bei längeren Zeiträumen stellen sich jedoch Alterungserscheinungen ein, die den Meßwert deutlich verfälschen. Bild 6.15 zeigt das Blockschaltbild eines Konversionsverfahrens, das Langzeitinstabilitäten weitgehend eliminiert.

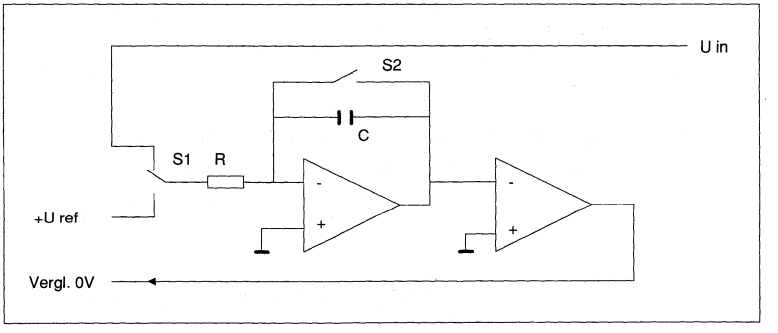


Bild 6.15: Schaltbild eines ADCs nach dem Doppelrampenverfahren

Nach dem Entladen des Kondensators über dem Integrator wird für eine konstante Zeitdauer die zu messende Eingangsspannung am Integrator angelegt. Am Ausgang des Integrator erscheint dadurch eine linear abfallende Rampe. Der Anstiegswinkel der Rampe hängt von der Eingangsspannung ab. Sobald die konstante Integrationszeit verstrichen ist, bringt der Schal-

ter S1 eine konstante, negative Referenzspannung an den Integrator, der damit eine ansteigende Rampe erzeugt (Dual Slope). Die Zeit, bis der Ausgang des Integrators wieder 0 Volt erreicht, hängt von der Höhe des Integrationswerts aus dem ersten Schritt ab. Diese Zeit ist damit proportional zur messenden Eingangsspannung. Die Langzeitstabilitäten des Widerstands und des Kondensators gehen nicht mehr in den Konversionswert ein, da sie sich in den aufeinanderfolgenden Rampen herausmitteln. Auch die konstante erste Integrationsdauer braucht nicht extrem genau zu sein, wenn man einen Taktgenerator mit nachgeschalteten Zähler benutzt, der auch als Zeitmesser für die zweite Rampe dient. Lediglich die Referenzspannung muß sehr stabil sein.

Die Dual-Slope-Methode kann in einfacher Weise abgewandelt werden, um auch bipolare Größen zu messen. Hierzu benötigt man nur eine zweite Referenzspannung, die genau den negativen Wert der Grundreferenz hat. Je nachdem, in welche Richtung der Integrator nach der Integration der Meßspannung läuft, führt man die Entladung mit der umgekehrten Referenz durch und zeigt die Polarität neben dem Digitalwert an (Bild 6.16).

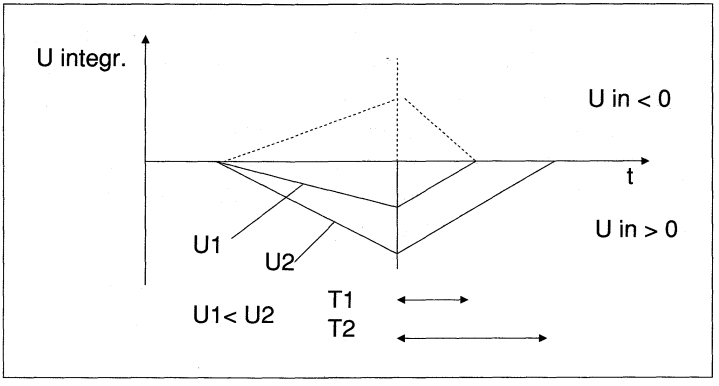


Bild 6.16: Beim Doppelrampenverfahren lassen sich sowohl positive als auch negative Eingangsspannungen konvertieren

6.7 Die Methode der sukzessiven Approximation

Bei allen Rampenverfahren hängt die Konversionszeit von der Eingangsspannung ab. Bei hohen Eingangsspannungen muß man deshalb lange Konversionszeiten hinnehmen. Rampenverfahren benötigen in der Regel einige Millisekunden zur Konversion. Beim Doppelrampenverfahren sind bei hoher Auflösung auch Konversionszeiten bis in den Sekundenbereich keine Seltenheit. Durch die in Bild 6.17 vorgestellte Schaltung läßt sich eine extrem schnelle Konversion erreichen. Man ersetzt dabei den Zähler durch einen Zwischenspeicher

(Latch) und versucht, die Größe der Eingangsspannung zu erraten. Dieser Ratevorgang darf natürlich nicht willkürlich geschehen, sondern erfolgt nach einem Wägeverfahren, bei dem man sich in wenigen Schritten an den Eingangswert herantastet. Zu Beginn der Messung wird das Latch mit einem Wert gefüttert, der genau der halben möglichen Maximalspannung entspricht. Der Komparator gibt danach sofort Auskunft, ob der Eingangswert in der oberen oder unteren Hälfte des Spannungsbereichs liegt. Befindet sich die Eingangsspannung in der unteren Hälfte, teilt man diesen Bereich erneut in zwei gleich große Teile auf, indem man ein Viertel des Maximalwerts am DAC erzeugt. Liegt der Eingangswert dagegen in der oberen Hälfte, so wird dieser Wert auch in zwei Teile aufgeteilt, indem man drei Viertel des Maximalwerts am DAC erzeugt. Jeder neue Versuch stellt sich auf das Ergebnis des letzten Versuchs ein und grenzt damit die Eingangsspannung immer mehr ein. Während man im schlimmsten Fall beim Rampenverfahren 256 Schritte benötigt, um eine 8-Bit-Genauigkeit zu erreichen, braucht das Verfahren nach der sukzessiven Approximation nur noch acht Schritte.

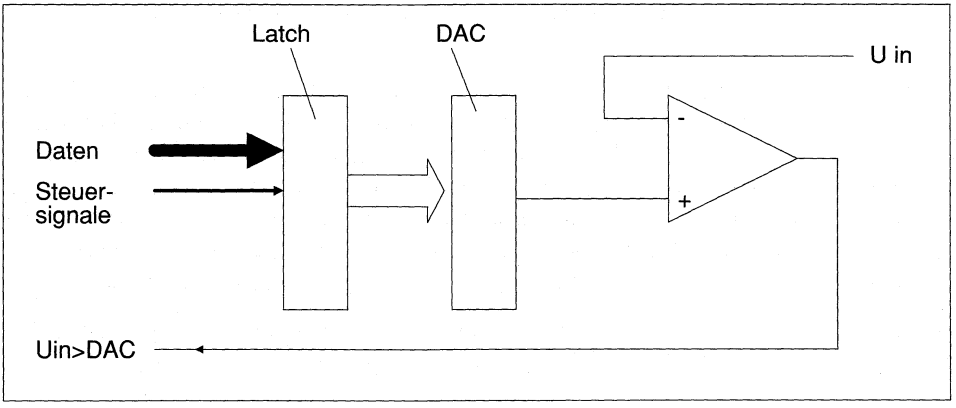


Bild 6.17: Prinzipieller Aufbau eines ADCs nach der sukzessiven Approximation.

Die Konversionszeit ist von der Eingangsspannung unabhängig.

Mit jedem zusätzlichen Schritt wird die Genauigkeit der Auflösung doppelt so hoch, während beim Rampenverfahren mit einer Verdoppelung der Auflösung auch eine Verdoppelung der Konversionszeit verbunden ist. Dafür müssen alle Komponenten des ADC eine hohe Genauigkeit aufweisen. Sowohl die kurzzeitigen als auch die langzeitigen Schwankungen wirken sich direkt auf die Konversionsgröße aus. Darüber hinaus versagt das Verfahren der sukzessiven Approximation immer dann, wenn die Eingangsspannung während der Konversion schwankt.

ADCs der sukzessiven Approximation enthalten deshalb häufig eine vorgeschaltete Logik, die den Eingangsspannungswert während der Konversion konstant hält. Eine derartige Schal-

tung wird Abtast-Halte-Glied (Sample and Hold) genannt, sie ist im Bild 6.18 dargestellt und besteht aus zwei Operationsverstärkern, die als Spannungsfolger geschaltet sind. Zwischen den Operationsverstärkern befindet sich ein Umschalter, der mit einem Kondensator verbunden ist.

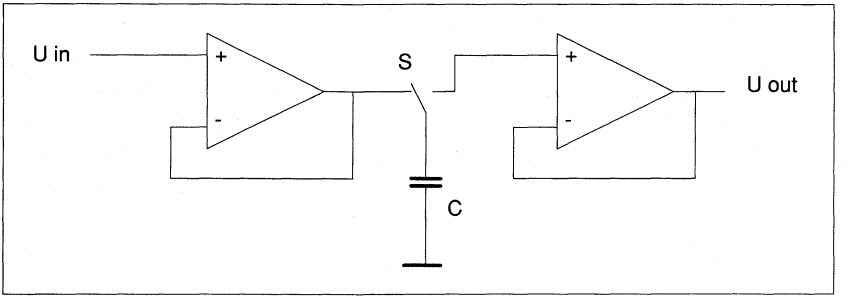


Bild 6.18: Schaltbild eines Abtast-Halte-Gliedes (Sample and Hold)

Vor dem Meßbeginn lädt der Schalter den Kondensator auf den Momentanwert der Eingangsspannung auf. Nach dem Umschalten erscheint die Momentangröße am Ausgang des zweiten Operationsverstärkers. Jetzt kann die Konversion beginnen. Da der Eingang des zweiten Operationsverstärkers keinerlei Stromfluß benötigt (in Wirklichkeit ist der Stromfluß extrem klein: 1 bis 100 pA), verändert die Kondensatorspannung während der gesamten Messung ihren Wert nicht. Nach erfolgter Konversion wird der Schalter wieder in seine Ausgangslage zurückgestellt und der Kondensator wird mit der Eingangsgröße geladen.

Im praktischen Teil dieses Buches befinden sich zahlreiche Anwendungen, die auf das Rampenverfahren und die Methode der sukzessiven Approximation eingehen.

6.8 Spezielle Konversionsverfahren

Abschließend seien noch zwei wichtige Spezialverfahren zur Digital-Wandlung vorgestellt. Das erste Verfahren konvertiert eine zu messende Eingangsspannung in einem einzigen Schritt und ist als Parallel-ADC (Flash-ADC) bekannt. Bild 6.19 zeigt den Aufbau eines Parallel-ADC.

Eine Referenzspannung wird durch eine Widerstandskette gleicher Widerstände in äquidistante Spannungswerte aufgeteilt. Jeder Spannungswert wird dem Eingang eines Komparators zugeführt und mit dem zu messenden Eingangswert verglichen. Alle Komparatoren, die an den Referenzwerten liegen, die kleiner sind als die Eingangsspannung, zeigen einen positiven Ausgangswert. Die verbleibenden Komparatoren haben negatives Ausgangspotential.

Die Eingangsspannung wird damit auf einen Schlag konvertiert. Die digitale Ausgangsinformation zeigt dabei jeweils ein Datenwort, dessen Datenbreite der Konversionsgröße entspricht. Dieser Nachteil läßt sich jedoch durch die Verwendung eines nachgeschalteten ROMs zur Datenkodierung umgehen. Pro möglichem Schritt der Konversionsgröße wird jeweils ein Widerstand und ein Komparator benötigt. Für geringe Auflösungen (z.B. 16 Schritte; also 4 Bit) sind Parallel-ADCs besonders geeignet. Sie lassen sich durch die heutige Integrationstechnik gut auf einem Chip unterbringen und garantieren dann eine gute Linearität.

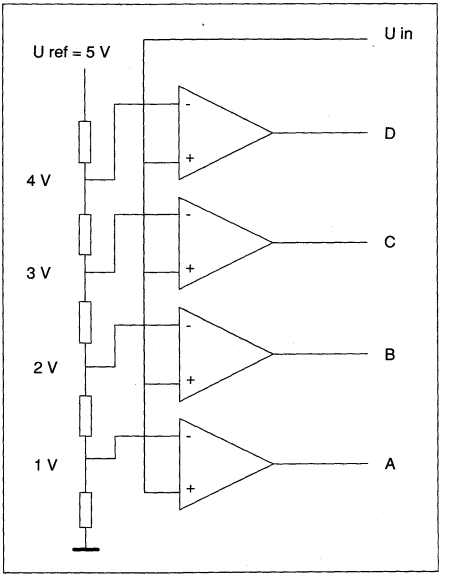


Bild 6.19: Prinzipieller Aufbau eines Parallel-ADCs (Flash-ADC)

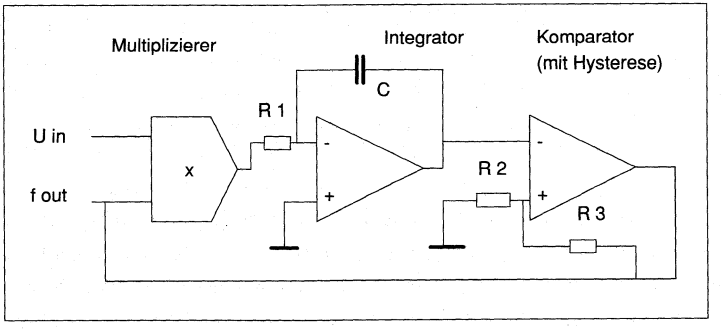


Bild 6.20: Schaltbild eines Spannungs-/Frequenz-Wandlers

Ein weiteres Verfahren ist dann sinnvoll, wenn die Analogwerte galvanisch zu trennen oder über große Distanz zu übertragen sind. Es handelt sich hierbei um die Spannungs-/ Frequenz-Wandlung. Kernstück des ADC ist ein Oszillator, der eine Frequenz erzeugt, die zu der zu messenden Eingangsspannung streng proportional ist. Bild 6.20 zeigt den Aufbau eines Spannungs-/Frequenz-Wandlers. Er besteht aus einem Multiplizierer, einem Integrator und einem Komparator mit Hysterese. Der Multiplizierer erzeugt an seinem Ausgang das Produkt aus dem digitalen Wert des Komparators und dem Wert der Eingangsspannung. Der Integrator bildet damit an seinem Ausgang eine linear ansteigende Rampe, die in ihrem Anstiegswinkel der Eingangsspannung entspricht. Sobald der Hysteresepunkt des Komparators erreicht ist, schaltet dieser um, und der Integrator ändert die Richtung der Rampe (Bild 6.21).

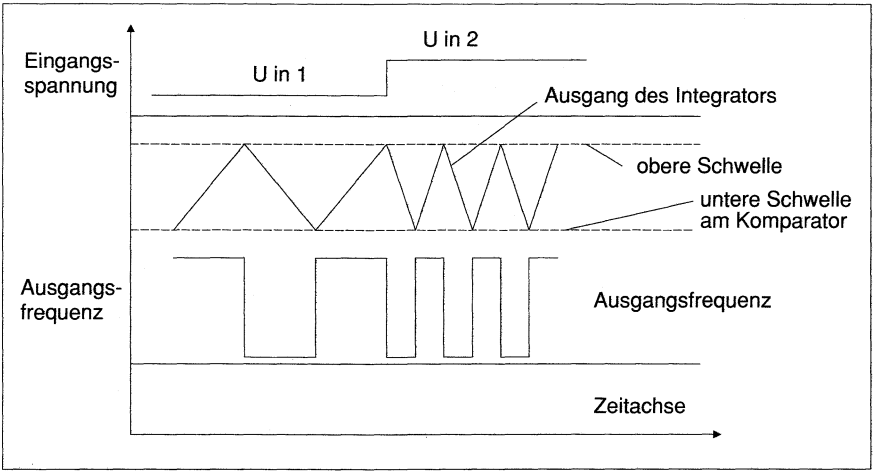


Bild 6.21: Signalverlauf beim Spannungs-/Frequenz-Wandler

Die Zeit, in der die Hysteresespannung des Komparators erreicht wird, hängt direkt von der Eingangsspannung ab. Damit erhält man am Ausgang des Komparators eine Frequenz, die genau der Eingangsspannung entspricht. Dieses Verfahren benötigt zur Konversion nur noch einen Frequenzzähler, der über einen Zeitraum die Impulse des Komparators auswertet. Impulse oder Frequenzen lassen sich über einen nahezu beliebig langen Weg transportieren, während der entsprechende Analogwert nach diesem Transport nur in stark verfälschter Form ankäme. Darüber hinaus gelingt die galvanische Trennung über einen einzigen Optokoppler, unabhängig davon, wie viele Bits an Konversionsauflösung dem Verfahren zugrunde liegen. Schwankende Eingangsspannungen stören den Spannungs-/Frequenz-Wandler fast nicht. Es besteht sogar die Möglichkeit, aus einer verlängerten Frequenzmessung (mit langer Torzeit) den wirklichen Mittelwert der Eingangsspannung zu bestimmen. Allen diesen Vorteilen steht der Nachteil einer relativ langen Umsetzzeit gegenüber. Konversionszeiten von einigen Millisekunden bis in den Sekundenbereich sind üblich.

6.9 Fehler der Konversionsverfahren

Digitale Meßgeräte vermitteln heutzutage oftmals den Eindruck, als sei der angezeigte Wert absolut genau. Die Anzeige mehrerer Stellen hinter dem Komma läßt den Benutzer glauben, er könne eine Spannung von 100 Volt und mehr auf weniger als ein Millivolt genau bestimmen. Doch hier ist eher das Gegenteil richtig. Die digitale Anzeige ist oft nur eine Folge der preiswerten Integration, wohingegen präzise Widerstände und driftarme Bauteile hohe Kosten verursachen. Eine genaue Konversion bis zur letzten Stelle ist in der Regel nur von guten und daher teueren Meßgeräten zu erwarten. Die Gründe für eventuelle fehlerhafte Messungen sind im folgenden näher erklärt:

Bild 6.22 zeigt die Konversion eines Analogwertes in einen Digitalwert bei idealer Konversion. Die Abweichung der analogen Spannung vom Digitalwert ist über die gesamte Konversionsgerade hinweg höchstens um die halbe Quantisierungseinheit falsch. (Der Fehler beträgt damit 1/2 LSB: Least Significant Bit.)

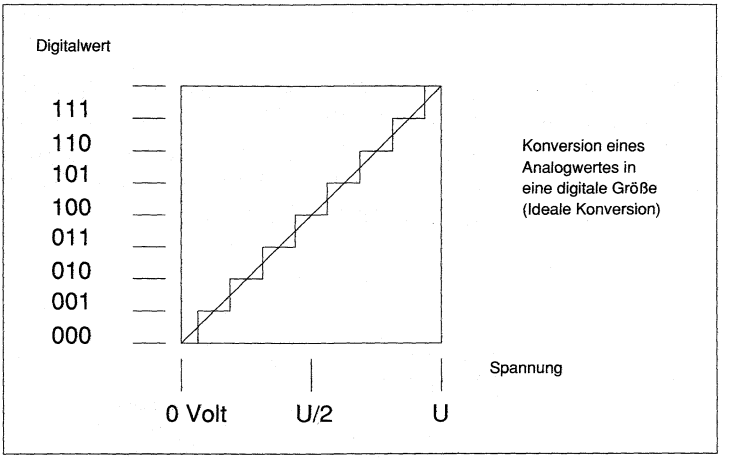


Bild 6.22: Bei der idealen Konversion weicht der tatsächliche Eingangswert höchstens um ein LSB (Least Signification Bit) vom Meßwert ab

Bei der Methode der sukzessiven Approximation treten vereinzelt Quantisierungsfehler auf, weil die Quantisierungssprünge nicht den Einzelteilspannungen entsprechen. Eine geringe Toleranz der Widerstände im DAC führt zu Fehlern der differentiellen Nichtlinearität, wobei die Bewertung der Schritte gegeneinander ungleich ist (Bild 6.23).

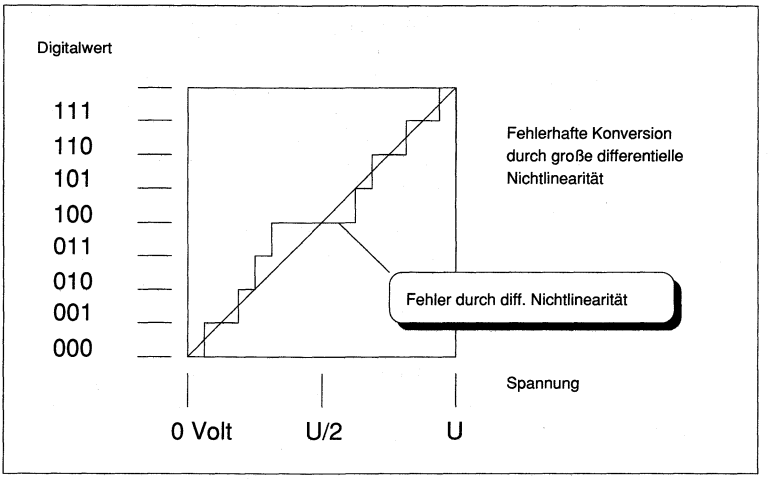


Bild 6.23: Fehler durch differentielle Nichtlinearität

Sobald die Summe zweier Viertelschritte um einen ganzen Quantisierungschritt größer (oder kleiner) als ein Halbschritt ist, entsteht eine sichtbare Lücke in der digitalen Zuordnung. Man erhält dann auf der Digitalseite einen Codefehler, da ein direkter Sprung von einer auf die übernächste Einheit stattfindet. Ändert sich die digitale Information über einen größeren analogen Bereich nicht, liegt ein Monotoniefehler vor (Bild 6.24).

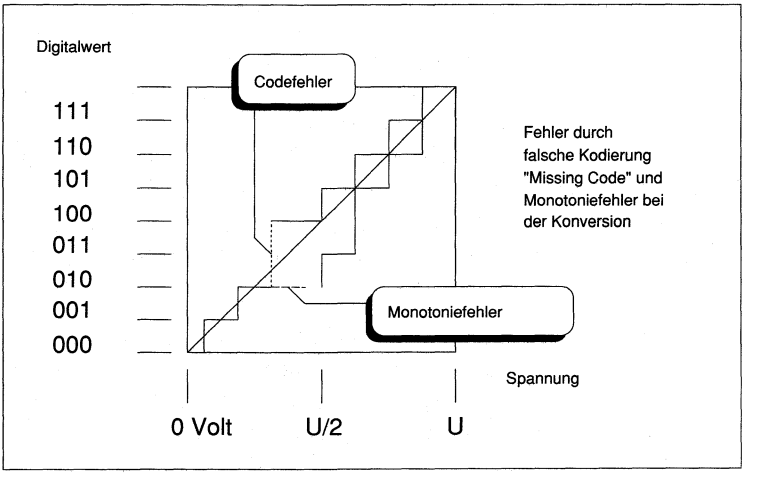


Bild 6.24: Code- und Monotoniefehler können bei der Digital-Wandlung nach der sukzessiven Approximation auftreten

Neben den Quantisierungsfehlern, die sich auf der Bitebene abspielen, können durch das Zusammenwirken aller Schaltungseinheiten noch zusätzlich Fehler auftreten, die ihre Wirkung auf der gesamten Konversionslinie haben. Fehler entstehen häufig, weil die Eingangsverstärker nicht die richtige Verstärkung zeigen oder die Referenzspannungen falsch bewertet sind. In diesen Fällen erhält man einen Verstärkungsfehler, der mit zunehmendem Eingangswert eine immer größere Abweichung zeigt (Bild 6.25).

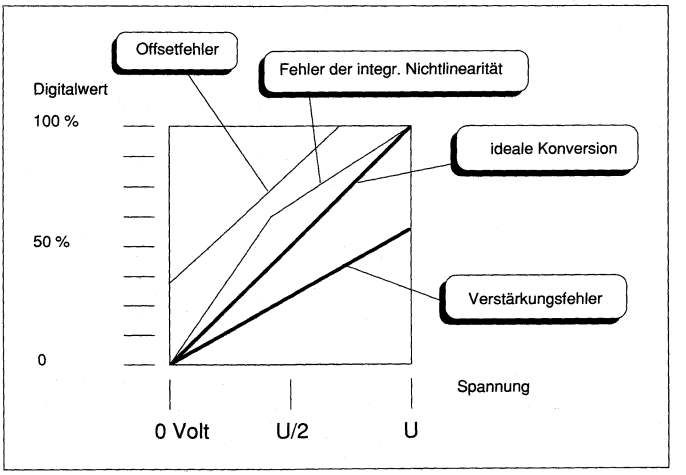


Bild 6.25: Fehler der Konversionslinie beim Zusammenwirken aller Komponenten im ADC

Sobald ein Verstärker nicht richtig auf den Nullpunkt abgeglichen ist, oder ein eventuell eingesetzter Komparator eine Differenzspannung zusätzlich zum Vergleichswert addiert, entsteht ein Offsetfehler. Alle konvertierten Werte sind in diesem Fall gegen den wahren Meßwert verschoben. Besonders tückisch sind Fehler durch eine integrale Nichtlinearität, die weder im Nullpunkt noch beim Spannungsmaximalwert in Erscheinung treten. Lediglich in unterschiedlichen Bereichen (z.B. in der Mitte) weicht die Konversionsgröße vom Meßwert ab. Derartige Fehler treten immer dann auf, wenn ein ADC oder ein DAC zwar richtig abgeglichen wurde, aber durch den internen Aufbau eine nichtlineare Kennlinie eines Elements enthält.

7 Programmieren mit dem Personalcomputer

Es war die Art zu allen Zeiten, durch Drei und Eins und Eins und Drei Irrtum statt Wahrheit zu verbreiten.
Goethe

7.1 Programmiersprachen im Überblick

Wie in den ersten Kapiteln dieses Buches erläutert, kann ein Computer auf der Hardwareebene nur Bitkombinationen verarbeiten. Ein bestimmtes Bitmuster veranlaßt die Zentraleinheit, gewisse Operationen auszuführen, die in ihrer Gesamtheit einen Assemblerbefehl ergeben. Wegen der einfacheren Darstellungsmöglichkeiten werden diese Bitkombinationen meist nur als Hexadezimalzahlen dargestellt. Exakt müßte es Sedezimalzahl heißen, aber jeder redet von Hexadezimal oder nur Hexzahl und so geschieht das auch hier. Hierbei gibt es eine eindeutige Zuordnung von der Bitkombination zur Hexzahl. Es ist lediglich eine andere, vereinfachte Schreibweise. Jedes Programm, ganz gleich in welcher Sprache es ursprünglich geschrieben wurde, muß in die Maschinensprache übersetzt werden, damit der Computer die Befehle verarbeiten kann. Das ist auch der Grund, warum die Programme von einem Computertyp nicht auf einem anderen ausgeführt werden können.

Am Anfang der Computerentwicklung wurden die Rechner tatsächlich in Maschinensprache programmiert, wobei in der einfachsten Form die Befehle Bit für Bit an Schaltern eingestellt werden mußten. Man kann sich vorstellen, wie schwierig und fehleranfällig eine solche Programmierung war. Andererseits kannten die Programmierer ihren Computer aufs Bit genau. Ein entscheidender Schritt wurde getan, als man jedem Maschinenbefehl ein für den Menschen besser zu behaltendes Kürzel gab, das den Sinn des Befehls verdeutlichen konnte. Zum Beispiel wird der Assemblerbefehl zum Addieren mit ADD, der zum Subtrahieren mit SUB abgekürzt. Dieser sogenannte mnemonische Code kann von einem Computer nicht mehr unmittelbar als Befehl verarbeitet werden, sondern muß erst wieder in die Maschinensprache zurückübersetzt werden. Ein dafür geeignetes Programm nennt man Assembler, die Gesamtheit der mnemonischen Befehle bilden die Assemblersprache, meist ebenfalls Assembler genannt. Diese Assemblersprache ist in der Hauptsache eine für Menschen besser geeignete Maschinensprache. Jedem Befehl der Maschinensprache entspricht genau ein Befehl der Assemblersprache.

Noch einen Schritt weiter geht man bei den höheren Programmiersprachen wie Basic, C, Fortran, Pascal, Prolog und wie sie alle heißen. Der Befehlsvorrat dieser Sprachen orientiert

sich nicht mehr an dem Computer, sondern an den Problemen, die damit bearbeitet werden sollen. Daher nennt man diese Sprachen auch problemorientiert, während die Assemblersprache maschinenorientiert ist. Die Befehle der höheren Programmiersprachen werden von einem Programm, das Compiler oder Interpreter genannt wird, in die entsprechenden Maschinenbefehle übersetzt. Im allgemeinen arbeitet ein Programm in einer höheren Programmiersprache langsamer als ein entsprechendes Assemblerprogramm, dafür hat man den Vorteil, in diesen Sprachen viel schneller und effektiver arbeiten zu können. Ein weiterer Vorteil für die professionellen Hersteller von Software besteht in der Möglichkeit, die Programme einer höheren Programmiersprache relativ einfach von einem Computertyp auf einen anderen zu übertragen. Da die Programme bei einer höheren Programmiersprache als Textdateien vorliegen, braucht nur diese Datei auf den neuen Computertyp transferiert und mit einem Compiler für diese Programmiersprache übersetzt zu werden. In der Regel sind zwar auch hierbei gewisse Änderungen an dem Programm erforderlich, doch ist der Zeitaufwand dabei viel geringer als bei einem Assemblerprogramm.

Bei vielen Sprachen sind dieser Portierbarkeit von Software enge Grenzen gesetzt, da die Sprachen in einer Unzahl von Dialekten vorkommen. Es gibt Normen zwar auch schon für Programmiersprachen, aber wenn sich die Hersteller nur an diese Normen halten würden, wären manche Sprachen nicht sehr komfortabel (z.B. würden alle Grafik-Befehle fehlen). Die am stärksten genormten Sprachen sind sicher Cobol und Ada, die aber im Mikrocomputerbereich keine entscheidenden Rollen spielen. Auch C soll weitgehend portabel sein. Bei Pascal ist das schon viel weniger der Fall und bei Basic überhaupt nicht. Bei den höheren Programmiersprachen gibt es prinzipiell zwei verschiedene Formen von Übersetzungsprogrammen: Interpreter und Compiler. Bei einem Interpreter wird das Programm der höheren Programmiersprache ohne eine vollständige Übersetzung in Maschinensprache direkt ausgeführt. Hierzu wird jeder einzelne Befehl des Ursprungsprogrammcodes im Speicher abgelegt und erst während der Verarbeitung in die Maschinensprache übertragen. Man nennt diesen Vorgang auch interpretieren; hierbei wird also kein Objektcode erzeugt. Dafür müssen sowohl der Interpreter als auch das Quellprogramm während der Ausführung im Computer vorhanden sein.

Interpreter haben vor allem während der Programmentwicklung Vorteile, da die Auswirkung einer Änderung im Programm nicht erst nach langwierigen Kompilier- und Linkvorgängen sichtbar wird, sondern unmittelbar nach dem Eintippen des Befehls. Gerade dieser Vorteil des Interpreters kann aber auch zu einem schlechten Programmierstil beitragen. Oft werden die Probleme vor dem Programmieren nicht mehr genügend durchdacht, sondern man probiert nach dem Verfahren Trial and Error (Versuch und Irrtum) solange herum, bis man eine halbwegs funktionierende Lösung gefunden hat. Interpreter sind vor allem für die Sprache BASIC gebräuchlich. Im Gegensatz hierzu wird bei einem Compiler das ganze Programm einer höheren Programmiersprache in die Maschinensprache übersetzt. Auf diese Weise erhält man ein lauffähiges Programm und der Compiler wird nicht mehr benötigt. Da die Übersetzung in Maschinensprache während des Programmablaufs nicht mehr notwendig ist, läuft ein kompiliertes Programm schneller ab als ein interpretiertes.

Hier noch ein Wort zu dem in diesem Buch verwendeten Basic-Dialekt. Wie bereits oben erwähnt, ist die Standardisierung bei Basic noch nicht sehr weit fortgeschritten. Für den Atari ST stellt – mindestens im deutschsprachigen Raum – GFA-Basic so etwas wie einen Standard dar, weswegen dieser Dialekt auch in diesem Buch verwendet wird. GFA-Basic hat viele Strukturelemente von moderneren Sprachen wie Pascal und C übernommen und zeichnet sich durch einen mächtigen Befehlssatz aus.

7.2 Schritte zur Programmentwicklung

Am Anfang eines Softwareprojekts – egal wie groß oder klein es ist – steht eine unklare Vorstellung über die Leistungen des zu erstellenden Programms. Bei vielen Projekten nehmen diese unklaren Vorstellungen erst während der Programmentwicklung eine konkrete Gestalt an. Ja, sehr häufig überlegt sich der Progammierer erst während er den Programm-Code schreibt, was das Programm überhaupt leisten soll und wie die dafür notwendigen Algorithmen aussehen müssen. Diese Vorgehensweise könnte man als Chaos- oder Hau-Ruck-Methode bezeichnen und es sollte unmittelbar einleuchten, daß man so eigentlich nicht arbeiten kann. Man stelle sich einmal vor, ein Haus würde auf diese Weise gebaut, ohne einen genauen Plan, ohne Architekten, nur nach den verschwommenen Angaben des Bauherrn – unvorstellbar. Und doch wird in vielen Firmen – selbst bei kleinen Softwarehäusern immer noch so gearbeitet.

Das Ergebnis ist die sogenannte Softwarekrise. Während die Preise für Hardware immer weiter verfallen, nimmt der Aufwand und damit die Kosten für die Entwicklung und Wartung der Software immer mehr zu. Will man die oben beschriebene Chaos-Methode vermeiden, so braucht man eine Strategie, die aus der unklaren Vorstellung (des Auftraggebers) ein sauber strukturiertes Projekt macht. Ein Teil dieser Strategie muß darin bestehen, das Gesamtprojekt in überschaubare Schritte zu unterteilen, die nacheinander ausgeführt werden können. Die Gesamtfunktionen des Programms sollten in Teilfunktionen aufgegliedert werden, die dann zu Funktionsgruppen, den sogenannten Modulen zusammengefaßt werden können. Das Ziel dieser Modularisierung besteht darin, große Teile eines Projekts auf allgemein vorhandene Funktionen zu reduzieren. Das Rad muß schießlich nicht immer wieder neu erfunden werden.

Man kann die hier skizzierten Schritte in sogenannte Phasen zusammenfassen und nennt das Vorgehen danach auch Phasenkonzept. Eine solche Phase ist durch eine bestimmte Tätigkeit charakterisiert, für die bestimmte Hilfsmittel, Methoden und Werkzeuge zur Verfügung stehen. Darunter sind Werkzeuge für die eigentliche Programmierung zu verstehen, wie Editor, Compiler, Linker, Debugger, aber auch solche Dinge wie Fragenkataloge, Checklisten, Testformulare u.ä. Eine, aber bei weitem nicht die einzige, mögliche Einteilung solcher Phasen sieht folgendermaßen aus:

Phase 1: Analyse der unklaren Vorstellung: Lastenheft
Phase 2: Definition des Projektumfangs: Pflichtenheft
Phase 3: Projektkonstruktion (Definition aller Module)
Phase 4: Modulfertigung
Phase 5: Systemintegration, Tests und Freigabe
Phase 6: Installation beim Endkunden
Phase 7: Wartung

Die Dokumentation ist nicht als eigene Phase aufgeführt, weil diese Tätigkeit jede Phase begleiten muß. In der Phase 1 muß die unklare Vorstellung konkretisiert werden. Bei komerziellen Projekten haben die Kunden häufig unrealistisch hohe Anforderungen an ein Programm. Das Ziel dieser Phase ist ein Anforderungskatalog für das Projekt, hier Lastenheft genannt. Für diese Definition sind zunächst keine Computerkentnisse erforderlich.

In der Phase 2 wird das Lastenheft in eine Programmbeschreibung umgesetzt, die hier Pflichtenheft heißt. Darin ist schon beschrieben, wie das Programm die im Lastenheft gestellten Aufgaben bewältigen soll. Hier sollten die benötigten Datenstrukturen ebenso beschrieben sein wie die Benutzerschnittstelle (z.B. Menüs und Masken) des Programms. Für ein umfangreiches Projekt ist ein solches Pflichtenheft einige hundert Seiten stark. Dieses Pflichtenheft sollte sich der Auftraggeber genau durchlesen, da es die Grundlage für alle weiteren Arbeiten darstellt. Nachträgliche Änderungen sind meist mit erheblichen Kosten verbunden.

In der Phase 3 werden die im Pflichtenheft noch in natürlicher Sprache beschriebenen Funktionen aus Unterprogrammen zusammengesetzt. Die Leistungsmerkmale jedes dieser Unterprogramme wird genau beschrieben. Hier müssen nun die allgemeine Probleme des Pflichtenhefts genau durchdacht werden. Häufig stellt man in dieser Phase noch Widersprüche oder Unklarheiten im Pflichtenheft fest. In dieser Phase müssen auch die Algorithmen entworfen werden, mit denen die beschriebene Leistung implementiert werden soll. Unter einem Algorithmus ist das Umwandeln eines Problems in seine mathematischen und logischen Abläufe gemeint. Ein Hilfsmittel hierbei sind Flußdiagramme (Bild 7.1) oder Struktogramme, da hiermit ein komplexer logischer Zusammenhang übersichtlich dargestellt werden kann. In einem Flußdiagramm wird der Lösungsweg (Algorithmus) mit Hilfe genormter grafischer Symbole dargestellt. Grundlage dieser Symbole ist die Norm DIN 66001 »Sinnbilder für Datenfluß- und Programmablaufpläne«, die im Anhang abgebildet ist.

In der Phase 4 beginnt nun die eigentliche Programmierung, d.h. das Kodieren aller beschriebenen Unterprogramme. Zu jedem Unterprogramm muß eine Testumgebung geschrieben werden, durch die sichergestellt wird, daß alle in Phase 3 beschriebenen Leistungsmerkmale auch richtig erfüllt werden.

In der Phase 5 müssen alle Teile des Programms zusammengefaßt und insgesamt getestet werden. Die Ergebnisse der Tests sollten immer in einem Testprotokoll festgehalten werden.

In der Phase 6 wird das fertige Produkt beim Kunden installiert und das Bedienpersonal eingewiesen. Jetzt zeigt sich, wie gut das Programm an der bisherigen Arbeitsweise des Kunden

angelehnt ist. Hier geht es um die Akzeptanz der Mitarbeiter an eine neue Arbeitsweise. Gerade in dem Bereich Messen, Steuern, Regeln, wo eine technische Betrachtung vorherrscht, werden die damit verbundenen Probleme häufig übersehen.

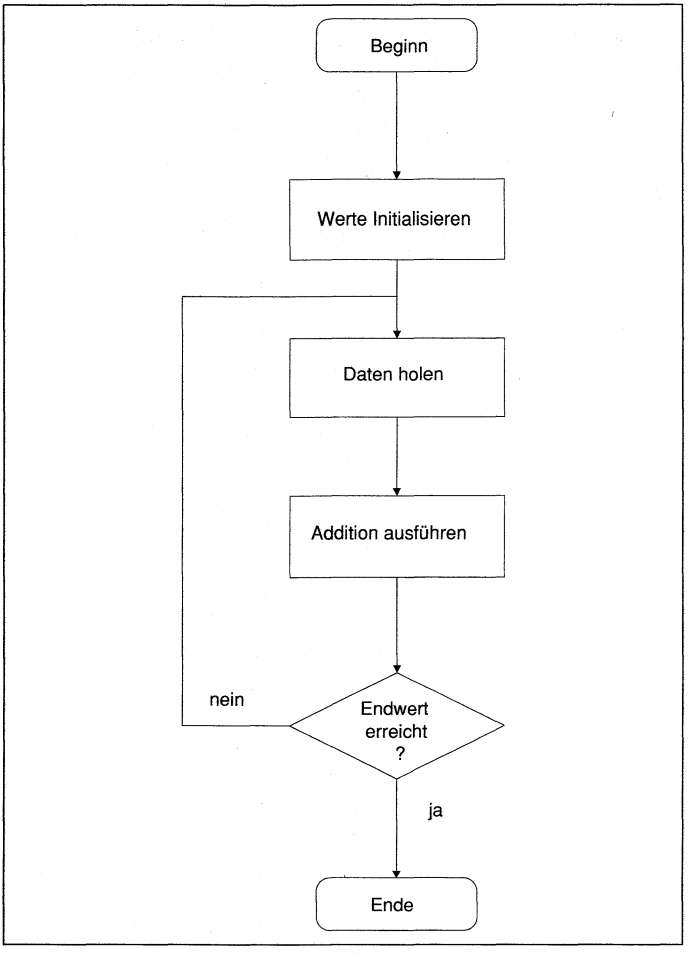


Bild 7.1: Einfaches Beispiel für ein Flußdiagramm

Die letzte Phase umfaßt die Benutzung und Wartung des Programms. Gerade in dieser Phase macht sich ein gut strukturiertes und modularisiertes Programm bezahlt, da damit die unvermeidlichen Änderungs- und Erweiterungswünsche (hoffentlich) leicht implementiert werden können.

7.3 Programmiermodelle des 68000

Dieses Kapitel beschäftigt mit der Assemblerprogrammierung des 68000. Der interne Aufbau des Prozessors wird dabei nur soweit behandelt, wie dies für die Programmierung notwendig ist. Die allgemeinen Hardware-Eigenschaften wurden bereits in Kapitel 4.2 beschrieben. In Bild 7.2 ist die Registerstruktur des 68000 dargestellt. Diese Register sind Speicher innerhalb der CPU, die durch Maschinenbefehle angesprochen werden können. So kann z.B. ein Datum vom externen Speicher in das Register D0 der Zentraleinheit kopiert werden. Der 68000 verfügt über acht allgemeine Daten-Register, sieben Adreß-Register, zwei Stapelzeiger (USP, User Stack Pointer und SSP, Supervisor Stack Pointer), einem Befehlszeiger (PC, Program Counter) und einem Statusregister (SR, Status Register). Alle Register und Statusregister sind 32 Bit breit.

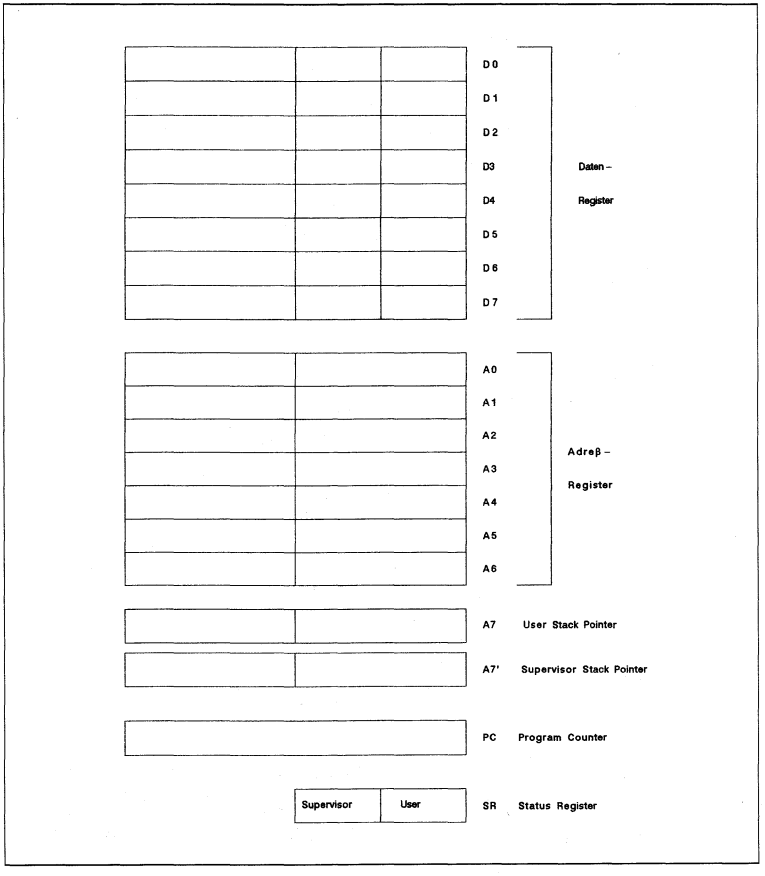


Bild 7.2: Die Registerstruktur des 68000

Damit die folgenden Erläuterungen jeweils mit Beispielen aus der Assemblersprache des 68000 verdeutlicht werden können, soll hier ein Befehl aus der Gruppe der Datentransferbefehle vorgezogen werden, der sehr häufig verwendet wird: der Move-Befehl (move = verschieben, bewegen). Mit diesem Befehl werden Daten von einer Quelle zu einem Ziel kopiert, er hat folgende allgemeine Form:

MOVE Quelle, Ziel

Die einfachste Form des MOVE-Befehls kopiert den Inhalt eines CPU-Registers in ein anderes, z.B.:

MOVE D0, D1

Dieser Befehl kopiert den Inhalt von Register D0 in das Register D1. Die Bezeichung MOVE ist irreführend, denn die Daten werden nicht verschoben, sondern tatsächlich kopiert. Nach der Ausführung des Befehls stehen also im Register D0 und D1 dieselben Inhalte. Die Richtung des Datentransfers ist durch die Reihenfolge der Register im Argument des Befehls angegeben. Die Daten werden in das Register kopiert, das im Argument an zweiter Stelle steht. Um die Daten in die umgekehrte Richtung zu transportieren, also vom Register D1 ins Register D0, muß der Befehl MOVE D1, D0 lauten. Dieser MOVE-Befehl funktioniert sowohl für die Daten- wie für die Adreßregister, z.B.

MOVE D1, A1
MOVE A1, A7

Für viele Anwendungen benötigt man gar nicht die ganze Länge der Register, sondern nur einen Teil davon. Dies ist z.B. bei den Zeichen des Alphabets der Fall. Diese belegen jeweils nur ein Byte und es wäre sehr unpraktisch, dafür immer 4 Byte verwenden zu müssen. Bei den Registern spielt das noch keine Rolle, aber irgendwann müssen die Daten auch in den Speicher kopiert werden. Daher unterstützt der 68000 bei den Datenregistern folgende Datentypen:

Datentyp	**Länge**	**Zahlenbereich**
Bit	1Bit	1
BCD	4 Bit = 1/2 Byte	15
Byte	8 Bit = 1 Byte	255
Wort	16 Bit = 2 Byte	65.535
Langwort	32 Bit = 4 Byte	4.294.967.299

In Bild 7.2 sind die Typen Byte, Wort durch senkrechte Striche angedeutet.

Für die Datentypen Bit und BCD stehen in der Assemblersprache spezielle Befehle zur Verfügung. Die anderen Datentypen Byte, Wort und Langwort können bei fast allen Befehlen

verwendet werden, bei denen Daten transferiert werden. Bei diesen Anweisungen muß dann aber auch ausdrücklich angegeben werden, auf welchen Datentyp sie sich beziehen:

Datentyp	Befehlsformat bei MOVE
Byte	MOVE.B
Wort	MOVE.W oder MOVE
Langwort	MOVE.L

Ohne eine Typangabe beziehen sich die Befehle also auf ein Wort.

Die Adreßregister sind ebenfalls 32 Bit breit, wovon aber nur die ersten 24 Bit benutzt werden, da der Prozessor nur über 23 Adreßleitungen verfügt. Bei den Adreßregistern können nur die Datentypen Wort und Langwort verwendet werden, aber es können auch Daten in Adreßregistern gespeichert werden.

Wie in den Kapitel 1 und 2 beschrieben, muß der Mikroprozessor jeden Befehl zunächst aus dem Speicher holen (Opcode-Fetch) bevor er ausgeführt werden kann. Der Befehlszeiger dient der internen Programmverwaltung und zeigt jeweils auf die Speicherstelle, die den nächsten Befehl enthält.

Der 68000 kann in zwei verschiedenen Modi arbeiten, dem Supervisor Mode (Betriebssystem-Modus) und dem User Mode (Benutzer-Modus). Wie die Namen schon andeuten, ist der Supervisor Mode für das Betriebssystem reserviert, während alle Anwenderprogramme im User Mode laufen. Diese beiden Modi werden auch von der Assemblersprache unterstützt. So gibt es beim 68000 privilegierte Befehle, die nur im Supervisor Mode ausgeführt werden können (diese Befehle werden im nächsten Kapitel angegeben). Auch das Umschalten zwischen den beiden Betriebsarten wird durch bestimmte Befehle ermöglicht.

Diese strikte Trennung hat einige Vorteile, so kann z.B. im User Mode auf die Daten des Betriebssystems nicht zugegriffen werden, wodurch sie vor versehentlichem Überschreiben geschützt sind.

Die Unterscheidung zwischen diesen beiden Modi spiegelt sich auch in der Registerstruktur wieder. So ist das Adreßregister A7, der sogenannte Stackpointer, doppelt vorhanden. Damit haben das Betriebssystem und die Anwenderprogramme jeweils verschiedene Stackbereiche. Diese beiden Register werden von der CPU je nach Betriebsart automatisch umgeschaltet, ohne das sich der Programmierer darum kümmern müßte. Im User Mode ist es unmöglich, an den Inhalt des Supervisor SP zu gelangen, während es im Supervisor Mode einen Befehl gibt, um den Inhalt des User SP zu erhalten.

Auch das 16 Bit breite Statusregister spiegelt den Unterschied zwischen den verschiedenen Betriebsarten des 68000 wieder: Es ist in ein Statusregister für den User Mode und in eines für der Supervisor Mode unterteilt. Hierbei gehören die ersten 8 Bit von B0 bis B7 zum User

Mode, die oberen 8 Bit von B8 bis B15 zum Supervisor Mode. Im Usermode ist es zwar möglich, das Supervisor-Status-Byte zu lesen, nicht aber, es zu verändern. Im Supervisor Mode hat man natürlich den vollen Zugriff auf beide Teile des Statusregisters. Bild 7.3 zeigt den Aufbau des Statusregisters.

Wie der Name schon sagt, wird im Statusregister der Zustand der CPU wiedergegeben. Es gibt viele wichtige Befehle, die diese Zustandsbits auswerten. In einem Programm müssen z.B. Entscheidungen getroffen werden, die häufig vom Ausgang eines Vergleichs oder anderer Operation abhängen. Es soll z.B. eine Schleife so lange durchlaufen werden, bis der Inhalt eines Registers, das bei jedem Schleifendurchlauf heruntergezählt wird, den Wert 0 erreicht hat. Das Statusregister unterstützt solche Entscheidungen, indem beim Ausgang bestimmter Operationen eine Flagge (ein Bit im Statusregister) gesetzt oder zurückgesetzt wird. In dem angeführten Beispiel wird eine Flagge (das Zero-Bit) gesetzt, wenn der Schleifenzähler den Wert Null erreicht.

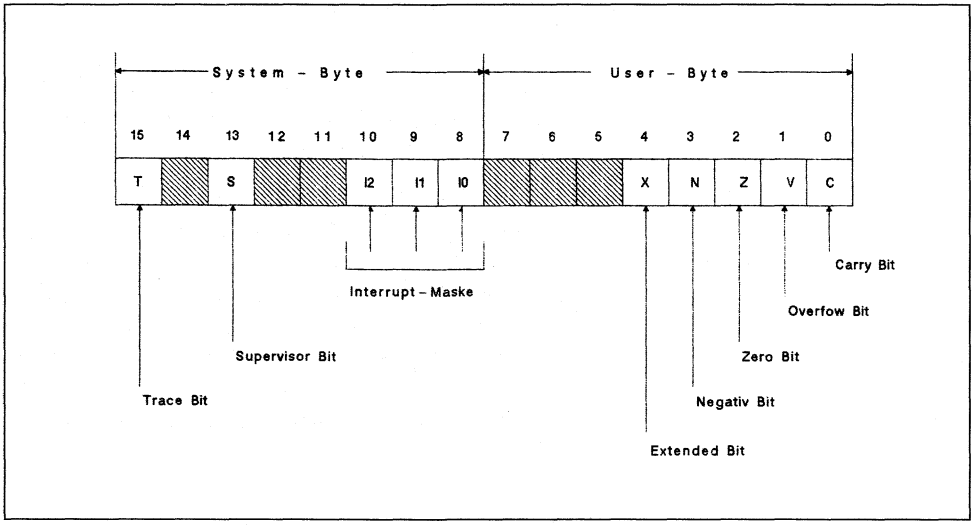


Bild 7.3: Aufbau des Statusregisters beim 68000

Im einzelnen enthält dieses Register folgende Bits:

Bit 0 — Carry – Übertrag: wird auf 1 gesetzt, wenn bei einer arithmetischen Operation ein Überlauf entsteht, sonst auf 0. Auch bei den Rotierbefehlen und bei der Multiplikation wird es beeinflußt.

Bit 1 — Overflow – Überlauf: wird auf 1 gesetzt, wenn bei arithmetischen Operationen der Datenbereich überschritten wird.

Bit 2 — Zero – Zero: wird auf 1 gesetzt, wenn das Ergebnis einer Operation gleich Null ist, sonst auf 0.

Bit 3 — Negativ – Negativ: wird auf 1 gesetzt, wenn das Ergebnis einer arithmetischen, logischen, Verschiebe- oder Rotieroperation negativ wird.

Bit 4 — Extend – Erweiterung: Dieses Bit ist eine Besonderheit des 68000. Es verhält sich im wesentlichen wie das Carry-Bit, wenn es gesetzt ist. Es wird aber z.B. bei den Vergleichsbefehlen (CMP) nicht gesetzt.

Bit 10 bis 12 — Interrupt-Maske – Unterbrechungsmaske: Mit diesen drei Bits lassen sich die maskierbaren Interrupts des 68000 ausblenden. Die Interruptstruktur ist in Kapitel 4.2 beschrieben.

Bit 13 — Supervisor – Betriebssystem: Mit diesem Bit werden die beiden Betriebsarten des 68000, Supervisor Mode (Betriebssystem Modus) und User Mode (Anwender Modus), unterschieden. Eine Eins bedeutet Supervisor Mode, eine Null User Mode.

Bit 15 — Trace – Steuert den Einzelschrittmodus. Damit ist es möglich, den 68000 per Software im Einzelschritt laufen zu lassen. Steht dieses Bit auf Eins, so springt der 68000 nach jedem Befehl einen festen Vektor an.

In diesem Kapitel wurde schon einmal das Adreßregister A7 (auch SP, Stack Pointer, Stapelzeiger genannt) angesprochen, mit dem der 68000 eine Stack, Stapel- oder auch Kellerspeicher genannte Datenstruktur unterstützt, die zum Zwischenspeichern von Daten besonders geeignet ist. Ein solcher Stapel kann mit einem Kistenstapel verglichen werden, wie in Bild 7.3 illustriert. Bei einem Stapel Kisten kann immer nur die oberste weggenommen werden. Analog kann man nur das zuletzt auf den Stapel abgelegte Datenwort wieder zurückholen. Eine neue Kiste bzw. eine neues Datenwort wird oben auf den Stapel hinzugefügt. Man nennt einen solchen Speicher auch FILO (First In Last Out, Zuerst hinein, zuletzt heraus).

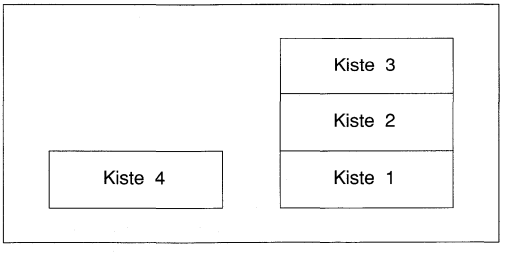


Bild 7.4: Stack-Prinzip am Beispiel eines Kistenstapels

Bei einem Computer ist der Stack ein Bereich des Hauptspeichers, der für diesen Zweck reserviert wird. Der Stack Pointer, beim Atari ST also das Adreßregister A7, zeigt immer auf

die aktuelle Stelle in diesem Bereich. Am Anfang zeigt der Stackpointer auf die höchste Adresse des Stack-Bereichs. Soll ein neues Datenwort auf den Stack gebracht werden, so muß zunächst der Stack Pointer erniedrigt werden und dann das neue Wort an die Stelle kopiert werden, auf die der Stack Pointer zeigt. Man nennt dies auch »einen Wert auf den Stack pushen« (push, stoßen). Der umgekehrte Vorgang, ein Wort vom Stack holen, heißt auch poppen (pop, abstoßen). Hierbei wird zuerst das Datum von der Adresse, auf die der Stack Pointer zeigt, geholt und anschließend der Stack Pointer erhöht (siehe Bild 7.5).

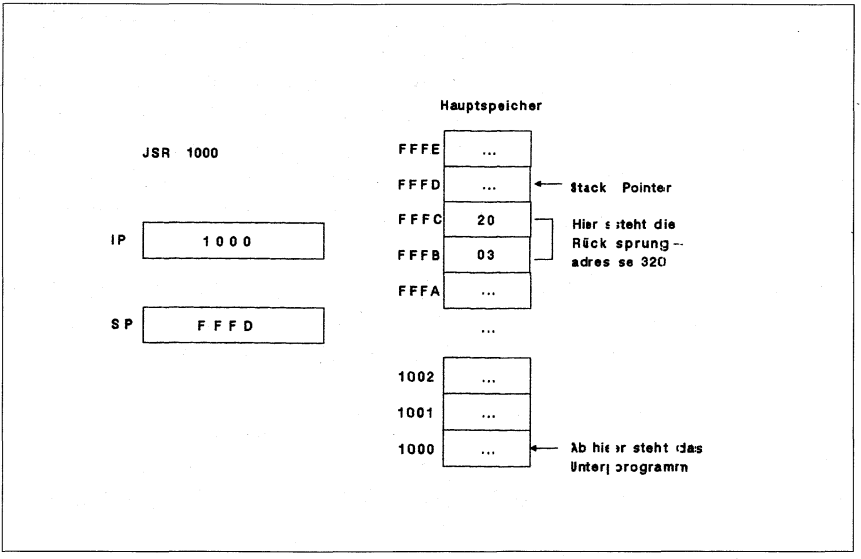


Bild 7.5: Kopieren von Daten auf den oder vom Stack

Zum kopieren von Daten zwischen der CPU und der Stack wird der MOVE-Befehl verwendet. Das genaue Aussehen dieses Befehls kann erst behandelt werden, wenn die verschiedenen Adressierungsarten des 68000 beschreiben wurden.

Der Prozessor benötigt aber auch für seine interne Verwaltung einen Stack. Bei Unterprogrammaufrufen oder Interrupts, springt der Prozessor an eine ganz andere Stelle des Hauptspeichers. Nachdem das Unterprogramm oder der Interrupt ausgeführt wurde, muß das Programm dann wieder an der richtigen Stelle fortgesetzt werden. Ja, es ist sogar eine Schachtelung von Unterprogrammen und Interrupts möglich. Zu diesem Zweck legt der Prozessor die Rücksprungadresse (d.h. die Adresse des nächsten Befehls nach dem Unterprogrammaufruf oder dem Interrupt) auf dem Stack ab, falls erforderlich auch mehrere. Nach dem Prinzip »zuerst hinein, zuletzt hinaus« können bei den entsprechenden Rückkehrbefehlen die Rücksprungadressen der Reihe nach wieder vom Stack geholt werden. Auch Parameterübergaben an Unterprogramme laufen häufig über den Stack.

Wie bereits erwähnt, verfügt der 68000 über zwei Stackpointer, einen für den Supervisor Mode und einen für den User Mode. Beide Stack Pointer liegen im Adreßregister A7. Mit dem Umschalten der Betriebsart wird auch der Inhalt des Registers A7 umgeschaltet.

7.4 Adressierungsarten beim 68000

Wie bereits im letzten Kapitel erwähnt, besteht ein großer Teil der Assemblerprogrammierung aus dem Hin- und Herschieben von Daten. Die Leistungsfähigkeit einer CPU zeigt sich daher auch darin, welche Möglichkeiten sie anbietet, die Quelle oder das Ziel dieser Daten zu bestimmen. Im allgemeinen werden die Daten von oder zu Adressen des Speichers kopiert. Die möglichen Arten, diese Adressen zu bestimmen, nennt man daher Adressierungsarten. Da als Quelle und Ziel von Daten auch die Register in einer CPU verwendet werden können, sind in den Adressierungsarten auch solche vorhanden, bei denen diese Register beteiligt sind.

Der 68000 bietet eine große Zahl verschiedener Adressierungsarten, die bei den meisten Befehlen, in denen Daten vorkommen, verwendet werden können. Man muß hier aber vorsichtig sein, da nicht bei allen Befehlen, alle Adressierungsarten für die Quelle und das Ziel der Daten erlaubt sind.

Die Adressierungsart geht immer aus dem Argument eines mnemonischen Befehls hervor. Das Programm zum Übersetzen der Assemblersprache in die Maschinensprache muß anhand der Schreibweise des Arguments die Adressierungsart erkennen und den Opcode für den Befehl festlegen.

Anhand der MOVE-Befehle sollen im folgenden die verschiedenen Adressierungsarten des 68000 erläutert werden:

- direkte Registeradressierung
- Absolute Adressierung
- Unmittelbare- oder Konstanten-Adressierung
- Adreßregister indirekt (ARI)
- Adreßregister indirekt mit Postinkrement
- Adreßregister indirekt mit Predekrement
- Adreßregister indirekt mit Adreßdistanz
- Adreßregister indirekt mit Adreßdistanz und Index
- PC-relative Adressierung
- PC-relative Adressierung mit Index

In den folgenden Bildern sind die bei der Befehlsausführung beteiligten Register und Speicherinhalte dargestellt. Pfeile symbolisieren bei den indirekten Adressierungsarten den Vektor, der auf die Daten in einer bestimmten Speicherstelle zeigt.

7.4.1 Direkte Registeradressierung

Bei dieser Adressierungsart werden die beteiligten Register direkt im Befehl angesprochen. Beim MOVE-Befehl werden die Daten dabei innerhalb der CPU von einem Register in ein anderes kopiert, der Speicher enthält nur den Maschinenbefehl für diese Operation und ist ansonsten nicht beteiligt, wie in Bild 7.6 gezeigt.

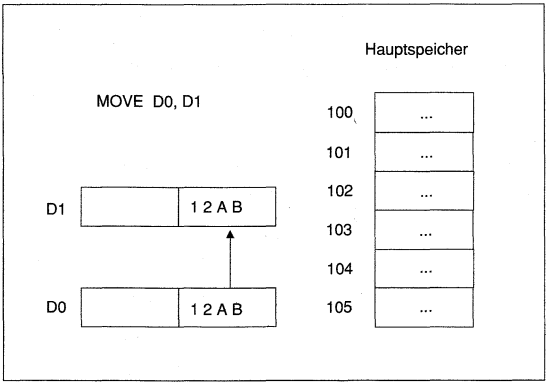


Bild 7.6: Registeradressierung

7.4.2 Absolute Adressierung

Bei der absoluten Adressierung ist die effektive Adresse des Datums direkt im Befehl enthalten. In dem Beispiel aus Bild 7.7 wird durch den Befehl MOVE 1000, D0 der Inhalt der Adresse 1000 in das Register D0 kopiert. Wichtig ist hierbei der Unterschied zur Konstanten-Adressierung, die im nächsten Abschnitt erläutert wird.

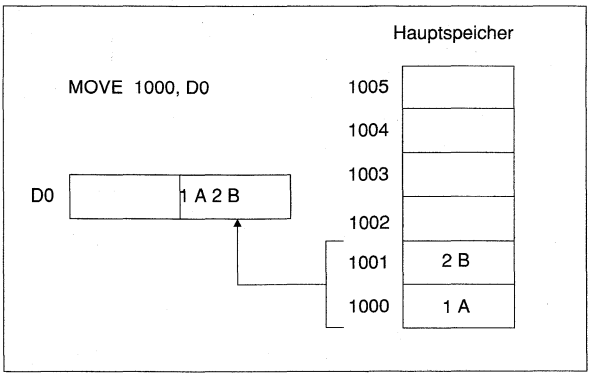


Bild 7.7: Absolute Adressierung

7.4.3 Unmittelbare oder Konstanten-Adressierung

Bei der unmittelbaren Adressierung ist das Datenwort als Konstante im Argument des Befehls enthalten. In dem Beispiel aus Bild 7.8 wird durch den Befehl MOVE #1000, D0 in das Register D0 die Konstante 1000 geschrieben. Bis auf das Zeichen "#" sieht der Befehl genauso aus wie bei der absoluten Adressierung, der Unterschied ist aber gravierend. Bei der Konstantenadressierung wird die Konstante 1000 in das Register kopiert, bei der absoluten Adressierung dagegen der Inhalt der Adresse 1000.

Ein "#" vor einer Zahl kennzeichnet beim 68000-Assembler eine Konstante.

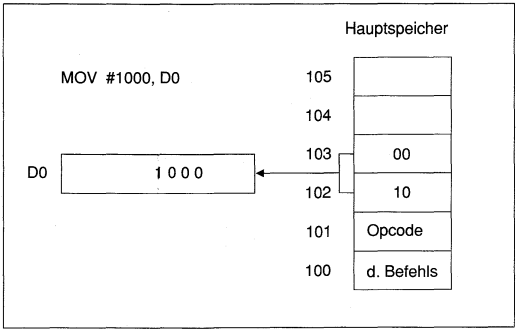


Bild 7.8: Unmittelbare Adressierung

7.4.4 Adreßregister indirekt (ARI)

Bei der indirekten Adreßregisteradressierung enthält das im Befehl angegebenen Register die Adresse, ab der sich die Daten befinden. Bei dem in Bild 7.9 angegebenen Befehl MOVE (A0), D0 enthält das Adreßregister A0 den Wert 14. Ab der Adresse 14 steht das Wort 1122, das durch den Befehl in das Ziel-Register D0 kopiert wird.

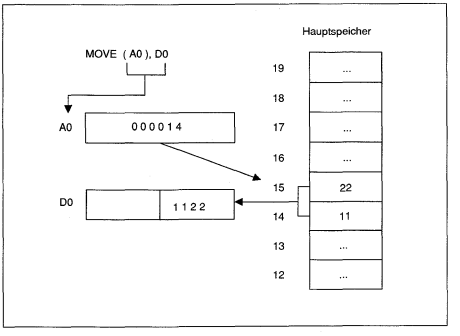


Bild 7.9: Adreßregister indirekt (ARI)

7.4.5 Adreßregister indirekt mit Postinkrement

Die indirekte Adreßregisteradressierung mit Postinkrement arbeitet genauso wie die indirekte Adreßregisteradressierung, nur wird hier nach dem Befehl das Adreßregister inkrementiert.

In dem Beispiel aus Bild 7.10 wird der Befehl zunächst genauso abgearbeitet wie in Bild 7.9. Die Daten werden also von derselben Speicherstelle geholt. Nach der Befehlsausführung steht im Adreßregister A0 aber nicht mehr der Wert $14, sondern $16.

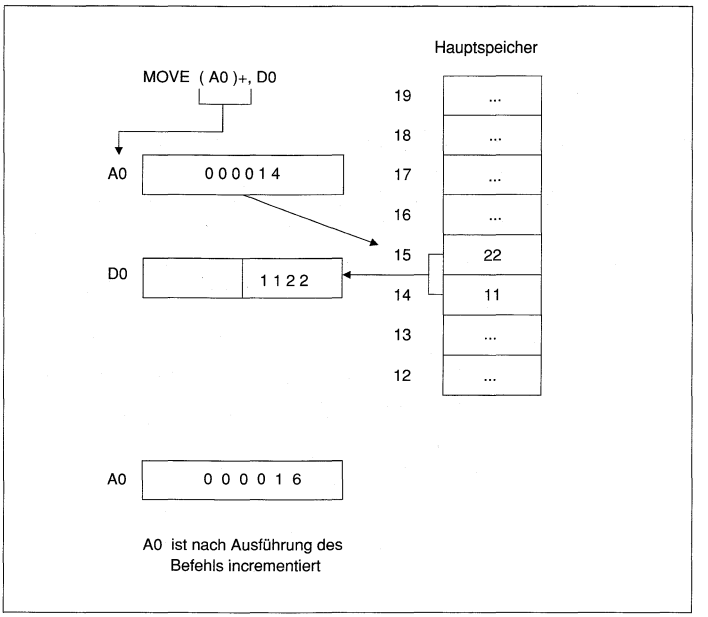


Bild 7.10: Adreßregister indirekt mit Postinkrement

Um welchen Wert das Adreßregister erhöht wird, hängt von der Datenlänge des Befehls ab:

Befehlsbeispiel		Beschreibung
MOVE.B	(A0)+, D0	Daten kopieren, dann A0 = A0 + 1
MOVE.W	(A0)+, D0	Daten kopieren, dann A0 = A0 + 2
MOVE.L	(A0)+, D0	Daten kopieren, dann A0 = A0 + 4

Mit diese Adressierungsart können also sehr einfach mehrere im Speicher hintereinander stehende Variablen bearbeitet werden.

7.4.6 Adreßregister indirekt mit Predekrement

Die indirekte Adreßregisteradressierung mit Predekrement arbeitet völlig analog zu der vorher beschriebenen, nur wird das Adreßregister nach dem Befehl nicht inkrementiert, sondern dekrementiert:

Befehlsbeispiel		Beschreibung
MOVE.B	-(A0), D0	Daten kopieren, dann A0 = A0 – 1
MOVE.W	-(A0), D0	Daten kopieren, dann A0 = A0 – 2
MOVE.L	-(A0), D0	Daten kopieren, dann A0 = A0 – 4

7.4.7 Adreßregister indirekt mit Adreßdistanz

Bei der indirekten Adreßregisteradressierung mit Adreßdistanz setzt sich die effektive Adresse, an der die Daten stehen, aus der Summe des Inhalts eines Adreßregisters und einer konstanten Adreßdistanz zusammen. Die Adreßdistanz wird im Befehl angegeben und ist eine vorzeichenbehaftete 16-Bit-Zahl (Bereich von – 32768 bis 32767).

In dem Beispiel aus Bild 7.11 beträgt die Adreßdistanz 100. Der Inhalt des Adreßregisters A0 ist 14 und die effektive Adresse demnach 114. Die Daten ab der Adresse 114, in Bild 7.11 also 1122, werden in das Datenregister D0 kopiert.

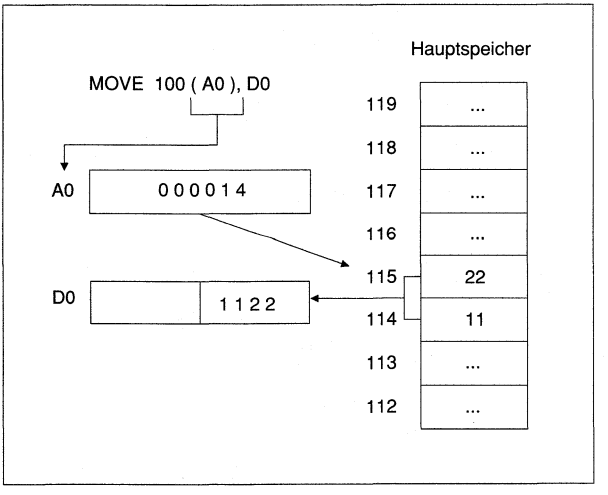


Bild 7.11: Adreßregister indirekt mit Adreßdistanz

7.4.8 Adreßregister indirekt mit Adreßdistanz und Index

Bei der indirekten Adreßregisteradressierung mit Adreßdistanz und Index setzt sich die effektive Adresse aus der Summe von drei Komponenten zusammen: Inhalt des Adreßregisters plus der im Befehl angegebenen konstanten Adreßdistanz plus dem Inhalt eines Indexregisters. Die Adreßdistanz kann in diesem Fall nur noch eine vorzeichenbehaftete 8-Bit-Zahl (Bereich von –128 bis 127) sein.

In dem Beispiel aus Bild 7.12 beträgt die Adreßdistanz 100. Der Inhalt der Adreßregister A0 ist 1000, der des Indexregisters A1 ist 20 und die effektive Adresse daher 1120 (Adreßdistanz + A0 + A1).

Beim Indexregister kann auch der Datentyp mit angegeben werden, z.B.:

```
MOVE.B    100(A0, A1), D0
MOVE.W    200(A3, A6), D5
MOVE.L    0(A5, A1), D8
```

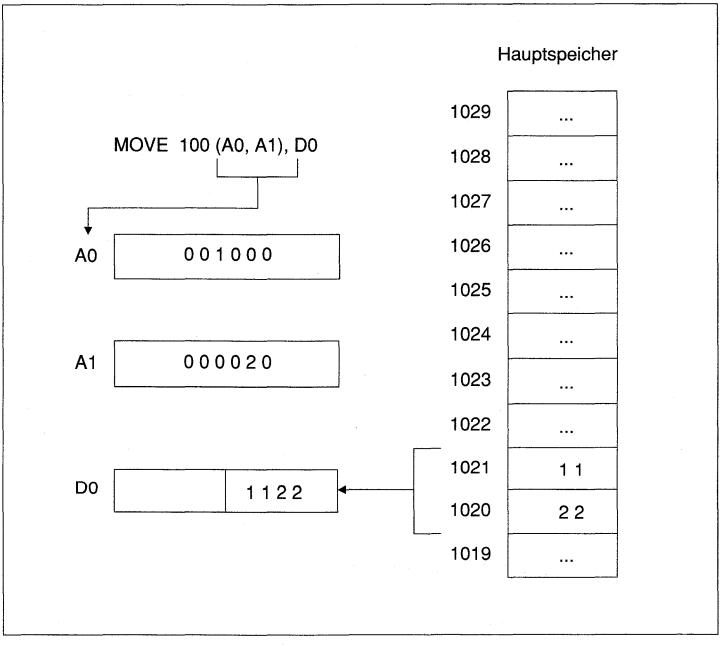


Bild 7.12: Adreßregister indirekt mit Adreßdistanz und Index

Diese Adressierungsart eignet sich vor allem zum Bearbeiten von Arrays.

7.4.9 PC-relative Adressierung

Bei der PC-relativen Adressierung setzt sich die effektive Adresse aus dem Inhalt des Program Counters (PC, Befehlszählers) und einer 16-Bit Konstanten (auch Offset genannt, Bereich von –32768 bis 32767) zusammen.

In dem Beispiel aus Bild 7.13 beträgt die Adreßdistanz 14, der Inhalt des PC 100 und die effektive Adresse demnach 114. Die Daten ab der Adresse 114 (in Bild 7.13 also 1122) werden also in das Datenregister D0 kopiert. Wird das Programm, das diesen Befehl enthält, an eine andere Stelle geladen, so ändert sich zwar der PC, die Programm-Daten stehen aber immer noch 1000 Byte hinter dem Befehl. Mit Hilfe dieser Adressierungsart können Programme ohne absolute Adressen geschrieben werden, die somit an jeder beliebigen Stelle des Speichers lauffähig sind.

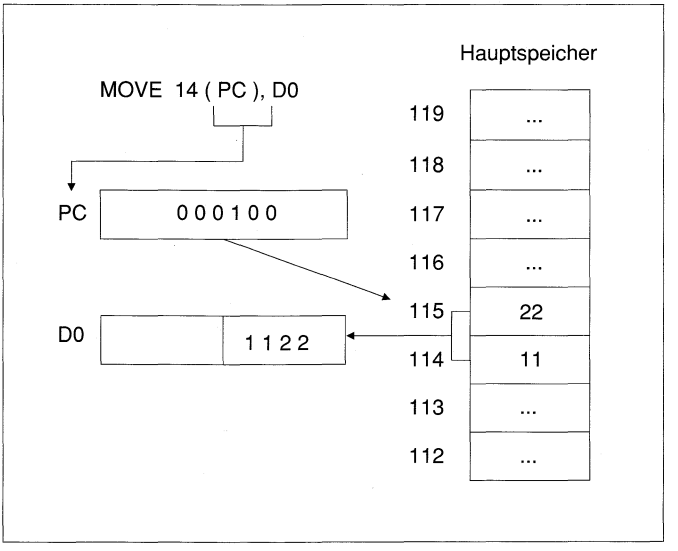


Bild 7.13: PC-relative Adressierung

7.4.10 PC-relative Adressierung mit Index

Die PC-relative Adressierung mit Index funktioniert analog zur indirekten Adreßregisteradressierung mit Adreßdistanz und Index (siehe Kapitel 7.4.8). Hier setzt sich die effektive Adresse ebenfalls aus drei Komponenten zusammen: dem Inhalt des PC, dem Offset und dem Inhalt des Indexregisters.

MOVE.L 1000(PC, A0), D0

Auch bei dieser Adressierungsart kann beim Indexregister ein Datentyp angegeben werden.

7.5 Die Assemblersprache des 68000 im Überblick

In diesem Kapitel soll nicht versucht werden, eine vollständige Einführung in den 68000-Asssembler zu geben. Das würde den Rahmen dieser Darstellung sprengen. Hier soll vielmehr ein erster Überblick gegeben werden, der es auch dem Leser ohne Assemblerkenntnisse erlaubt, die in Kapitel 8 vorgestellten Programme zu verstehen. In Kapitel 7.3 wurde die Registerstruktur und in 7.4 die Adressierungsarten des 68000 beschrieben. Hierauf wird im folgenden immer wieder zurückgegriffen. Es ist dabei sinnvoll, von Zeit zu Zeit Bild 7.2 noch einmal anzuschauen, damit sich die für das Programmieren in Assembler wichtigen Strukturen innerhalb der CPU einprägen. Der Befehlssatz eines Mikroprozessors wird mit jeder Generation immer komplizierter, weil immer mehr komplexe Befehle integriert werden und weil die internen Strukturen der Prozessoren immer umfangreicher wurden. Während der Befehlssatz der 8-Bit-Prozessoren noch sehr einfach und übersichtlich aufgebaut war, ist die Vielfalt der 68000-Befehle für einen Anfänger auf diesem Gebiet sicher sehr verwirrend. Doch im Vergleich zu anderen Prozessoren seiner Generation ist der Befehlssatz des 68000 noch sehr einfach und logisch aufgebaut und nach kurzer Einarbeitungszeit sicher auch für einen Anfänger der Assemblerprogrammierung (fast) mühelos zu beherrschen. In den folgenden Kapiteln werden die Befehle des 68000 in Gruppen vorgestellt.

Ein Mikroprozessor verarbeitet Daten. Das bedeutet aber nichts anderes als das Verknüpfen von Zahlen, ganz gleich, ob es sich dabei um Zeichen des Alphabets, Maschinenbefehle oder sonstige Daten handelt. Um dieses Verknüpfen zu ermöglichen, müssen die Daten zunächst aus einem externen Speicher in die CPU geholt, in der gewünschten Weise verarbeitet und falls erforderlich, wieder in den Speicher zurückgeschrieben werden. Ein Assemblerbefehl besteht aus Opcode (Abk. für Operation Code – Befehlscode) und Operand, wobei es einige Befehle gibt, die keine Operanden benötigen. Der Opcode legt den Charakter des Befehls fest und ist beim 68000 immer ein Wort lang. Dieser Charakter des Befehls wird auch durch das mnemonische Kürzel ausgedrückt. In den Operanden werden zusätzliche Informationen übergeben, die zur Ausführung des Befehls benötigt werden.

Beispiele

Maschinensprache		Assemblersprache	
Opcode	Operanden	mnem. C.	Argument
32 01		MOVE	D0,D1
30 36	12 AB	MOVE	#12AB, D0
52 40		ADDQ	#1,D0

Bei jedem neuen Befehl, den die CPU verarbeiten soll, wird zunächst das erste Wort aus dem Speicher geholt und dekodiert. Dieses erste Wort legt schon fest, ob es sich um einen Ein- oder Mehrwort-Befehl handelt. Besteht der Befehl nur aus einem Wort, kann er sofort ausgeführt werden. Im anderen Fall wird das zweite Wort aus dem Speicher geholt und, falls erforderlich, weitere eins bis vier Worte, bis alle Informationen zum Ausführen des Befehls vorhanden sind.

In der folgenden Übersicht sind alle Befehlsgruppen des 68000 zusammengestellt:

1. Befehle zum Datentransfer
2. Logische und arithmetische Befehle
3. Befehle zur Verwaltung der Zentraleinheit
4. Sprungbefehle
5. Interruptbefehle (Exceptions)

In den Kapiteln 7.4.1 bis 7.4.5 ist für jede Gruppe eine Liste der Befehle des 68000 aufgeführt. In diesen Listen wurden folgende Abkürzungen verwendet:

x	Angabe des Datentyps: .B, .W, .L,		
y	Angabe des Datentyps: .W, .L,		
Dn, Dm	eines der Datenregister D0, bis D8,		
An, Am	eines der Adreßregister A0, bis A7,		
adresse	eine Speicheradresse		
quelle	Quelle eines Datums, Speichers oder Registers,		
ziel	Ziel eines Datums, Speichers oder Registers,		
#konst	eine Konstante (muß immer mit "#" beginnen)		
cc	Condition Code, Bedingungscode,		
	Folgende Bedingungen sind erlaubt:		
	CC	Carry Clear, Carry gelöscht	C = 0
	CS	Carry Set, Carry gesetzt	C = 1
	EQ	Equal, gleich	Z = 1
	GE	Greater or Equal, größer oder gleich	N = V
	GT	Greater Than, größer als	N = V u. Z = 0
	HI	Higher, höher,	C = Z = 0
	LE	Less, kleiner	Z = 1 o. N V
	LS	Lower or Same, kleiner oder gleich	C = 1 o. Z = 1
	LT	Less Then, kleiner als	N V
	MI	Minus, minus	N = 1
	NE	Not Equal, ungleich	Z = 0
	PL	Positiv oder Plus,	N = 0
	VC	Overflow Clear, Überlauf gelöscht	V = 0
	VS	Overflow Set, Überlauf gesetzt	V = 1

Der Befehlssatz des 68000 kennt unterschiedliche Befehle für Operationen mit Daten- und Adreßregistern, z.B. MOVE für das Kopieren von Daten und MOVEA für Adressen. Dieser Unterschied geht aber auch aus dem Argument der Befehle hervor und eine Kennzeichnung im mnemonischen Code wäre gar nicht notwendig. Viele Assembler erlauben daher auch ein MOVE wo eigentlich ein MOVEA notwendig wäre. Dasselbe gilt für den ADD-Befehl und viele andere.

7.5.1 Datentransferbefehle

Die wichtigsten Befehle aus dieser Gruppe sind die MOVE-Operationen zum Kopieren von Daten, die bei der Behandlung der Adressierungsarten im Kapitel 7.4 als Beispiele dienten und daher hier nicht weiter erläutert werden müssen.

Befehl	Operand	Bedeutung
EXG	Rn, Rm	Inhalt von Registern austauschen
LEA	quelle, An	Laden eines Registers mit effektiver Adresse
LINK	An, #konst	Datenbereich auf dem Stack reservieren
MOVE.x	quelle, ziel	Kopieren von Daten
MOVE	quelle, CCR	Kopieren eines Bytes in das CCR-Register
MOVE	quelle, SR	Kopieren eines Worts in das Status-Register
MOVEA.x	quelle, An	Kopieren von Adressen
MOVEM.x	quelle, ziel	Kopieren mehrerer Register
MOVEP.y	quelle, ziel	Kopieren von Daten zur Peripherie
MOVEQ	#konst, Dn	Schnelles (Quick) Kopieren von Konstanten
PEA	adresse	Effektive Adresse auf den Stack legen
SWAP	Dn	Vertauschen der Worte eines Registers
UNLK	An	Datenbereich auf dem Stack freigeben

Der Befehl MOVEQ (Move Quick) ist nur ein Wort lang und daher in der Ausführung besonders schnell. Damit kann eine 8-Bit-Konstante in ein Register oder eine Speicherzelle kopiert werden. Zwei wichtige Befehle zur Unterstützung höherer Programmiersprachen sind LINK und UNLK, mit denen ein Bereich auf dem Stack reserviert bzw. freigegeben werden kann. Mit dem Befehl PEA (Push effective Adress) kann eine Adresse auf den Stack gelegt werden. Um ein Wort auf den Stack zu bringen, wird ebenfalls der MOVE-Befehl verwendet.

Ein typisches Beispiel ist der Aufruf von Betriebssystemfunktionen über TRAP #1 (dieser Befehl wird weiter unten besprochen). Hierbei werden auch die Parameter über den Stack übergeben. Das allgemeine Schema sieht dann folgendermaßen aus:

1. Parameter auf den Stack legen.
2. Funktionsnummer auf den Stack legen.
3. TRAP-Befehl aufrufen.
4. Stackpointer wieder in den Ausgangszustand bringen.

Diese Befehlsfolge soll am Beispiel der GEMDOS-Funktion Nr. 2 (ConOut, Ausgeben eines Zeichens an die Konsole) demonstriert werden:

```
move      #'x', -(sp)     ; das Zeichen x wird auf den Stack gelegt
move      #2, -(sp)       ; die Funktionsnummer ebenfalls
trap      #1              ; Aufruf von GEMDOS
addq.l    #4, (sp)        ; SP wieder korrigieren
```

Die ersten beiden MOVE-Befehle haben den SP um 4 erhöht (Stack-Operationen sind nur wortweise möglich), damit muß mit dem letzten Befehl wieder eine 4 addiert werden, um den SP auf den ursprünglichen Wert zu bringen.

7.5.2 Logische und arithmetische Befehle

Der 68000 verfügt über einen großen Vorrat an Befehlen, um logische Operationen, Vergleiche, Bitmanipulationen sowie mathematische Funktionen wie Addition, Subtraktion, Multiplikation und Division durchzuführen. Auch die Befehle für BCD-Arithmetik sind bereits implementiert.

Arithmetische Befehle

Befehl	Operand	Bedeutung
ABCD.B	quelle, ziel	Addition von BCD-Zahlen
ADD.x	quelle, ziel	Addition von Daten
ADDA.x	quelle, ziel	Addition von Adressen
ADDI.x	#konst, ziel	Addition von Konstanten
ADDQ.x	#konst, ziel	Schnelles (Quick) addieren einer Konstanten
ADDX.x	quelle, ziel	Addition mit Übertrag
CLR.x	ziel	Operand löschen
CMP.x	quelle, ziel	Vergleich zweier Daten
CMPA.x	quelle, ziel	Vergleich zweier Adressen
CMPI.x	#konst, ziel	Vergleich mit einer Konstanten
CMPM.x	(An), (Am)	Vergleich zweier Daten im Speicher
DIVS	quelle, Dn	Division mit Vorzeichen
DIVU	quelle, Dn	Division ohne Vorzeichen
EXT.x	Dn	Vorzeichenrichtige Erweiterung
MULS	quelle, Dn	Multiplikation mit Vorzeichen
MULU	quelle, Dn	Multiplikation ohne Vorzeichen
NBCD	ziel	Negation von BCD-Zahlen
NEG .x	ziel	Negation
NEGX.x	ziel	Negation mit X-Bit

Befehl	Operand	Bedeutung
SBCD	ziel, quelle	Subtraktion von BCD-Zahlen
SUB.x	quelle, ziel	Subtraktion von Daten
SUBA.x	quelle, An	Subtraktion von Adressen
SUBI.x	#konst, ziel	Subtraktion von Konstanten
SUBQ.x	#konst, ziel	Schnelles (Quick) subtrahieren von Konstanten
SUBX.x	quelle, ziel	Subtraktion mit X-Flag
TST.x	ziel	Operanden gegen Null testen

Bei allen arithmetischen und logischen Befehlen, deren Operand aus einer Quelle und einem Ziel besteht, wird die Quelle mit dem Ziel verknüpft und das Ergebnis in ziel abgespeichert: Quelle verknüpft Ziel -- ziel, zum Beispiel beim Addierbefehl:

ADD.B D0, D1

Hierbei werden die unteren 8 Bit der Register D0 und D1 addiert und das Ergebnis im Register D1 gespeichert. Wichtig sind noch die Quick-Befehle ADDQ und SUBQ, mit denen eine Konstante zu einem Register oder Speicherplatz addiert oder subtrahiert werden kann. Diese Befehle sind nur ein Wort lang und daher in der Ausführung sehr schnell. Die Konstante kann nur die Werte von 1 bis 8 annehmen. Diese Befehle werden vor allem zum Inkrementieren und Dekrementieren benutzt. Eine weitere wichtige, d.h. heißt häufig benutzte Gruppe sind die Vergleichsbefehle, z.B. CMP. Diese Befehle verändern die Operanden nicht, sondern das Ergebnis des Vergleichs wird im Statusregister angezeigt (Z bedeutet Zero Bit, C Carry Bit).

```
CMP  Operand_1,Operand_2
Operand_1 < Operand_2    <==>      Z = 0 und  C = 0
Operand_1 = Operand_2    <==>      Z = 1 und  C = 0
Operand_1 > Operand_2    <==>      Z = 0 und  C = 1
```

Nach einem Vergleich können die entsprechenden bedingten Sprung- oder Verzweigungsbefehle verwendet werden, um verschiedene Programmteile auszuführen.

Logische Befehle

Befehl	Operand	Bedeutung
AND.x	quelle, ziel	Logisches UND
ANDI.x	#konst, ziel	Logisches UND mit einer Konstanten
EOR.x	quelle, ziel	Logisches XOR
EORI.x	#konst, ziel	Logisches XOR mit einer Konstanten

Befehl	Operand	Bedeutung
OR.x	quelle, ziel	Logisches ODER
ORI.x	#konst, ziel	Logisches ODER mit einer Konstanten
BCHG	quelle, ziel	Bit umdrehen
BCLR	quelle, ziel	Bit löschen
BSET	quelle, ziel	Bit setzen
BTST	quelle, ziel	Bit prüfen
CHK	quelle, Dn	Datenregister gegen 0 und eine obere Grenze prüfen
TAS	ziel	Bit 7 testen und setzen

Schiebe- und Rotierbefehle

Befehl	Operand	Bedeutung
ASL.x	quelle, ziel	Arithmetisches Schieben nach links
ASR.x	quelle, ziel	Arithmetisches Schieben nach rechts
LSL.x	quelle, ziel	Logisches schieben links
LSR.x	quelle, ziel	Logisches schieben rechts
ROL.x	quelle, ziel	Links herum rotieren
ROR.x	quelle, ziel	Rechts herum rotieren
ROXL.x	quelle, ziel	Links herum mit X-Flag rotieren
ROXR.x	quelle, ziel	Rechts herum mit X-Flag rotieren

7.5.3 Befehle zur Verwaltung der Zentraleinheit

Befehl	Operand	Bedeutung
NOP		keine Operation
RESET		Rücksetzen der Peripherie (Reset-Leitung)
Scc	ziel	bedingtes Setzen eines Byte
STOP		Prozessor anhalten

In dieser Gruppe sind alle Befehle zusammengefaßt, die den Zustand des Prozessors in irgendeiner Weise beeinflussen. Hierbei sind die Befehle zum Setzen und Löschen von Bits im Statusregister, die eigentlich auch dorthin gehören, bereits bei den MOVE-Befehlen aufgeführt.

Auch der Befehl NOP (No Operation – Keine Anweisung), der nichts macht und nur CPU-Zeit verbraucht, gehört in diese Kategorie.

7.5.4 Sprungbefehle

Im normalen Betrieb bearbeitet der Prozessor die Befehle eines Programms in der Reihenfolge, in der sie im Speicher stehen. In praktisch jedem Programm müssen aber auch Verzweigungen enthalten sein, bei denen nach dem Ausgang bestimmter Entscheidungen verschiedene Programmteile ausgeführt werden (vergl. auch den CMP-Befehl). Um dies zu erreichen, bietet der 68000 zwei Befehlsarten an: Sprung- und Verzweigungsbefehle.

Die Verzweigungsbefehle sind immer relativ zum PC in einem Adreßbereich von +/– 32 Kbyte. Dagegen reichen die Sprungbefehle über den gesamten Adreßbereich.

Bei allen Sprungbefehlen wird die Ausführung des Programms an der im Befehl angegebenen relativen oder absoluten Adresse fortgesetzt.

Befehl	**Operand**	**Bedeutung**
Bcc	16-Bit-Offset	Bedingte Verzweigung
Bcc.S	8-Bit-Offset	Bedingte Verzweigung
BRA	16-Bit-Offset	Unbedingte Verzweigung
BRA.S	8-Bit-Offset	Unbedingte Verzweigung
BSR	16-Bit-Offset	Verzweigung zu einem Unterprogramm
BSR.S	8-Bit-Offset	Verzweigung zu einem Unterprogramm
DBcc	Dn,	Bedingte Schleife
JMP	ziel	unbedingter Sprung
JSR	ziel	Sprung zu einem Unterprogramm
RTR		Rückkehr von einem Unterprogramm mit Laden des Stausregisters
RTS		Rückkehr von einem Unterprogramm

Eine besonders wichtige Gruppe von Sprungbefehlen sind die Unterprogrammaufrufe BSR (Branch To Subroutine) und JSR (Jump To Subroutine). Hierbei wird die Programmsteuerung von einem Haupt- zu einem Unterprogramm verlegt. Am Ende des Unterprogramms stehen die RTR- oder RTS-Anweisung, mit der die Programmsteuerung wieder an das Hauptprogramm zurückgegeben wird.

Bei dem JSR-Befehl kann die Zieladresse durch verschiedene Adressierungsarten festgelegt werden. Bei Verwendung eines Assemblers werden die Sprungziele im Allgemeinen mit einem Namen versehen, so daß man sich um die Adreßberechnung nicht kümmern muß. Bei dem BSR-Befehl kann das Unterprogramm maximal 32 Kbyte vom Hauptprogramm entfernt liegen.

Die Unterprogrammtechnik trägt wesentlich dazu bei, auch ein Assemblerprogramm übersichtlich zu gestalten.

Bei diesen Unterprogrammaufrufen spielt der Stapelspeicher eine wichtige Rolle, da der Prozessor die Rücksprungadressen, bei der die Programmausführung nach dem Unterprogramm fortgesetzt werden muß, auf dem Stapel ablegt. Der Rücksprung-Befehl, der das Unterprogramm beendet, holt diese Adressen wieder vom Stapel ab.

Einige Beispiele für Verzweigungen und Sprünge:

```
BCS      F2          ; Verzweige, wenn C = 1.
BCC     A308         ; Verzweige, wenn C = 0.
BEQ      23          ; Verzweige, wenn Z = 1.
JMP     (A3)         ; Unbedingter Sprung, die Sprungadresse steht
                       im Register A3
JMP     100(PC,A4)   ;   " , die Sprungadresse berechnet sich aus
                       der Summe des PC, des Adreßregisters A4 und
                       des Offsets 100
JMP     12AB34       ;   " , die Sprungadresse steht als absolute
                       Adresse im Befehl
JSR     (A1)         ; Sprung zum Unterprogramm
```

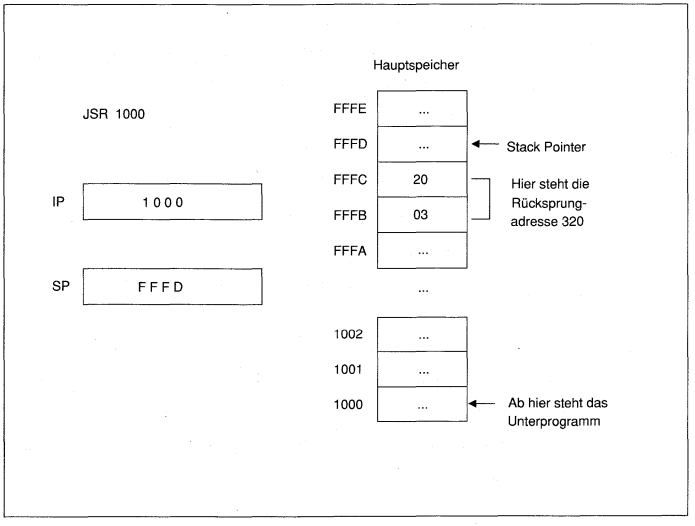


Bild 7.14: Der Befehl JSR 1000

7.5.5 Interruptbefehle

In Kapitel 5 wurde die Interruptstruktur des 68000 beschrieben und die Befehle zum Erzeugen eines Softwareinterrupts erwähnt, die nach der Motorola-Nomenklatur eigentlich Exception (Ausnahme) heißen.

Befehl	Operand	Bedeutung
RTE		Rückkehr von einem Softwareinterrupt (Exception)
TRAP	#n	Aufruf eines Softwareinterrupts (Exception) n ist eine Zahl zwischen 0 und 15
TRAPV		Aufruf eines Softwareinterrupts, wenn das V-Bit gesetzt ist
$A-Befehle		Line-A-Emulator
$F-Befehle		Line-F-Emulator

Der TRAP-Befehl ist so etwas wie ein Unterprogrammaufruf.

In der Vektortabelle der Interrupts sind für diesen Befehl 16 Einträge reserviert, die für jede der möglichen Nummern die Adresse des Unterprogramms enthalten.

Die Verarbeitung dieser Softwareinterrupts unterscheiden sich nicht von den durch die Hardware ausgelösten Unterbrechungen. Jedem der 16 möglichen Software-Interrupts ist in der Tabelle am Beginn des Speichers ein Vektor zugeordnet, der auf die Stelle zeigt, an der das Programm zum Bearbeiten des Interrupts steht. Jeder Eintrag in dieser Tabelle umfaßt 4 Byte. Den TRAP-Befehlen sind die Vektoren 32 bis 47 zugeordnet.

Daneben gibt es noch den TRAPV-Befehl, der nur dann ausgeführt wird, wenn das V-Bit (Overflow-Bit) gesetzt ist. Diesem Software-Interrupt ist der Vektor 7 zugeordnet. Damit kann eine einheitliche Behandlung von Überlauffehlern bei arithmetischen Operationen implementiert werden (wichtig beim Compiler-Bau).

Die letzten beiden Software-Interrupts sind die sogenannten Line-A- und Line-F-Emulatoren. Der Opcode für alle Maschinenbefehle ist beim 68000 ein Wort lang. Dabei entsprechen aber nicht alle möglichen 65535 Bitkombinationen auch einem gültigen Befehl. Trifft der Prozessor auf einen ungültigen Befehl, so wird normalerweise der Vektor Nr. 4 (illegaler Befehl) angesprungen. Hiervon ausgenommen sind zwei Gruppen von Bitkombinationen, die mit $A oder mit $F beginnen, und für die nochmals die beiden Vektoren 10 und 11 reserviert sind.

Praktisch wirkt also ein Opcode im Speicher, der mit $A oder $F beginnt, wie ein Softwareinterrupt (oder Unterprogrammaufruf) über die Vektoren 10 oder 11. Die übrigen 12 Bit können hierbei als Parameter verwendet werden.

Vek.-Nr.	Adresse	Funktion
7	$001C	Befehl TRAPV
10	$0028	Line A Emulator
11	$002C	Line F Emulator
32	$0080	TRAP #0
33	$0084	TRAP #1
.	.	.
.	.	.
47	$00BC	TRAP #15

Nach dem Auslösen eines Softwareinterrupts werden durch die CPU folgende Operationen durchgeführt:

- Der 68000 wird in den Supervisor Mode geschaltet.
- Der Inhalt des PC wird auf den Stack gelegt.
- Der Inhalt des Statusregisters wird auf den Stack gelegt.
- Der zum Trap gehörende Vektor wird in den PC geladen (entspricht einem Sprung zu dieser Adresse).

Jedes Unterbrechungsprogramm muß mit dem RTE-Befehl (Return from Exception, Rückkehr von der Unterbrechung) abgeschlossen werden. Nach diesem Befehl wird die Programmausführung mit der Anweisung nach dem Interruptbefehl fortgesetzt. Die Daten und Adreßregister werden nicht in den ursprünglichen Zustand zurückgesetzt. Hierfür muß der Anwender selbst sorgen. Der RTE-Befehl muß stets der letzte Befehl des Unterbrechungsprogramms sein.

Eine Programmiersprache kann man – genau wie eine natürliche Sprache – nur durch Üben richtig lernen. Nachdem die wichtigsten Befehle besprochen wurden, sollte der Leser alle Befehle selbst ausprobieren.

8 Aufbau und Anwendung der Interfacekarte

Der Worte sind genug gewechselt,
Laßt mich auch endlich Taten sehen.
Goethe

8.1 Bestückung und Test der Interface-Karte

Die zu dem Buch gehörende Interface-Karte dient als Grundlage zahlreicher Beispielschaltungen und Experimente, die mit dem Atari ST durchgeführt werden können. Die Karte erlaubt den problemlosen Einsatz in allen Atari-Typen, die über den ROM-Port verfügen. Sie besteht aus einer fest verdrahteten Logik, die als Port- und Businterface dient, und einem freien Feld zum Aufbau eigener Schaltungen.

Vor dem Bestücken der Platine sollten die richtigen Bauteile beschafft werden, die für die Grundfunktionen notwendig sind. Bild 8.1 zeigt diese Bauteile in einer geeigneten Bauform. Die folgende Stückliste enthält alle benötigten Teile und gibt einige Hersteller sowie mögliche Ersatztypen an:

Bauteil	Funktion	Hersteller (Vergleichstyp)
1 Platine	Gesamtaufbau	im Buch
2 IC-Sockel 20 polig		Fischer, AMP, Augat
2 IC-Sockel 14 polig		Fischer, AMP, Augat
1 IC-Sockel 16 polig		Fischer, AMP, Augat
1 IC-Sockel 40 polig		Fischer, AMP, Augat
1 IC-Sockel 28 polig		Fischer, AMP, Augat
1 SN 74LS374	Latch	TI, Motorola (74HCT374)
1 SN 74LS139	Decoder	TI, Motorola (74HCT139)
1 SN 74LS74	D-Flipflop	TI, Motorola (74HCT74)
1 SN 74LS125	Treiber	TI, Motorola
1 SN 74LS244	Treiber	TI, Motorola (74HCT244)
1 8255A	Par. Interf.	Intel, NEC (D71055C)
1 8251A	Ser. Interf.	Intel, NEC (8251C)
1 Stiftleiste 24 polig		Comatel, AMP

Bauteil	Hersteller (Vergleichstyp)
1 Stiftleiste 14 polig	Comatel, AMP
1 Leuchtdiode 3 mm (rot)	HP, Litronix
1 Widerstand 4K7 (1/4 Watt)	Vitrohm
1 Widerstand 1K (1/4 Watt)	Vitrohm
1 Kondensator 22 uF (25V)	Wima
1 Kondensator 0.1 uF (Keramik)	Wima
1 Kondensator 10 uF (25V) Tantal	Wima

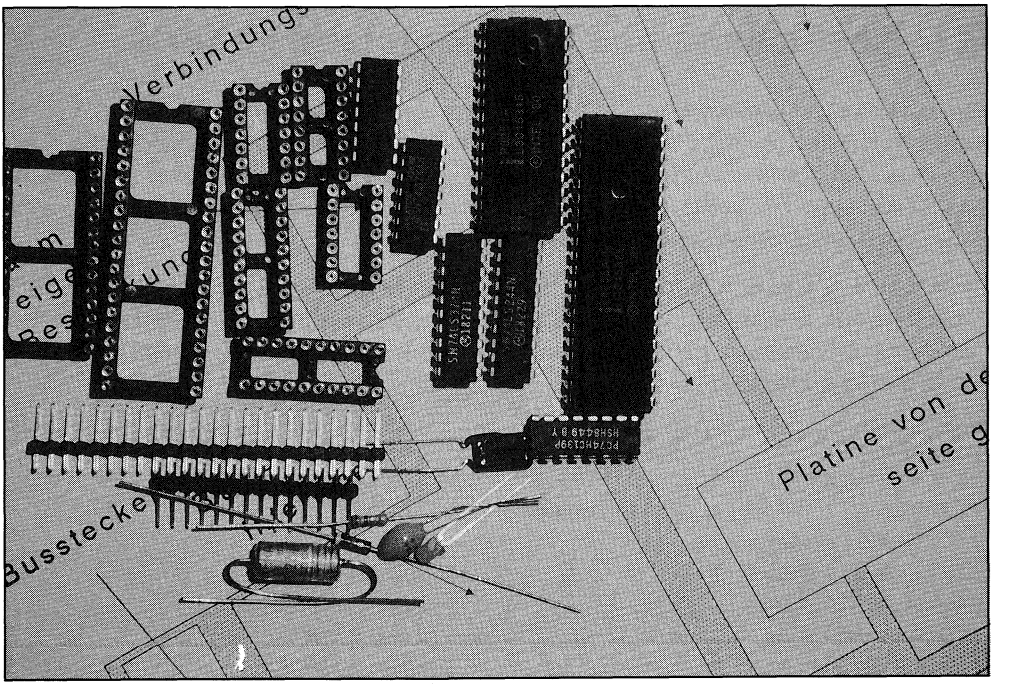


Bild 8.1: Bauteile zum Aufbau der Interface-Platine

Die Platine führt auf beiden Seiten Leiterbahnen, die jeweils in den Löchern eine elektrische Verbindung von der einen zu der anderen Seite haben. Sie ist mit einem Lack überzogen, der Lötzinn abstößt (Lötstopplack), was wesentlich zu einem sauberen Löten beiträgt. Die Oberseite der Platine ist in Bild 8.2 dargestellt und trägt darüber hinaus noch eine Aufschrift. Bestückungen dürfen nur von der Bestückungsseite vorgenommen werden. Die untere Seite (Lötseite) dient ausschließlich dem Anlöten der Bauteile. Auf der linken Seite der Platine

befindet sich der Interface-Busstecker, der zur besseren Kontaktierung vergoldet ist. Lötungen in diesem Bereich sind nicht zulässig. Wie Bild 8.2 zeigt, besteht die Platine aus sieben unterschiedlichen Bereichen:

- dem Busstecker auf der linken Seite
- dem Businterface neben dem Busstecker
- dem Verbindungsstecker
- dem Bereich der Peripheriebausteine
- dem Interfaceadapter
- und aus zwei Bereichen für eigene Bestückung

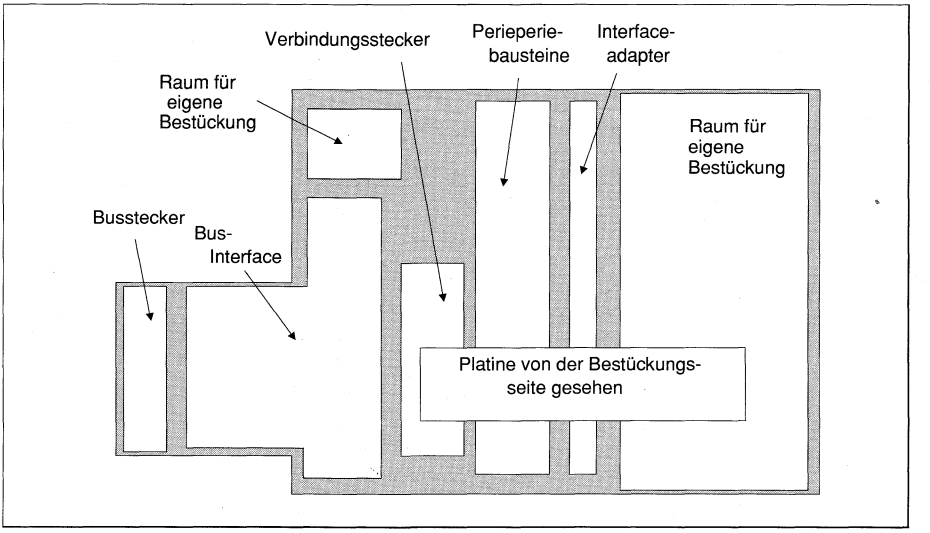


Bild 8.2: Die Interface-Platine mit den unterschiedlichen Bereichen

Die Platine kann in zwei unterschiedlichen Variationen aufgebaut werden, die dem entsprechenden Interface-Wunsch entgegenkommen. Bild 8.3 veranschaulicht den gesamten Platinenaufbau zum direkten Einstecken in den Atari ST. Den Bereich des Verbindungssteckers braucht man nicht zu bestücken, da alle notwendigen Leiterbahnen unterhalb des Steckers vorhanden sind. Die gesamte Platine wird in diesem Fall dicht am Rechner betrieben.

Eine weitere Modifikation der Platine zeigt Bild 8.4. Hier wird die Platine zwischen den beiden Verbindungssteckern durchgeschnitten und damit der Interfaceteil vom Peripherieteil getrennt. Man trennt die Platine zwischen den beiden Steckadaptern auf. Da nun alle Verbindungsleiterbahnen unterbrochen werden, muß man ein Flachkabel einsetzen, das die beiden Einheiten verbindet. Man hat damit die Möglichkeit, seinen eigenen Interfaceaufbau vom Atari ST räumlich zu trennen. Das Flachkabel sollte eine Maximallänge von 50 cm

jedoch nicht überschreiten, da sonst durch Reflexionen oder Übersprechen mögliche Fehlfunktionen zustande kommen.

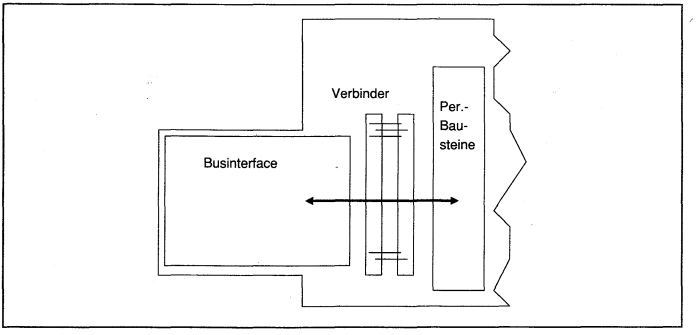


Bild 8.3: Der Bereich des ROM-Steckers (ohne Kabel)

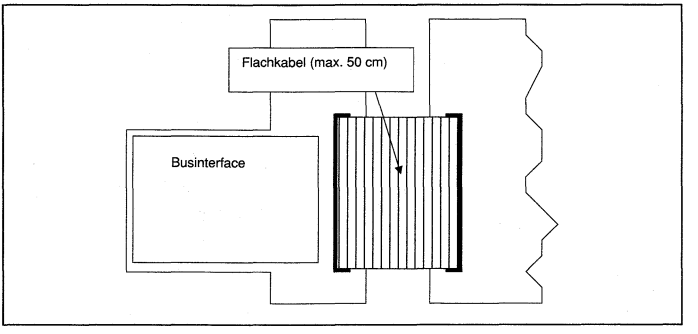


Bild 8.4: Der Bereich des ROM-Steckers mit Kabel bei zerschnittener Platine

Zwischen diesen beiden Varianten kann sich der Benutzer der Platine entscheiden. Ein späteres Umsteigen von der Variante 1 auf den Kabelausbau nach Variante 2 ist zwar möglich, aber nicht empfehlenswert, da sich die Platine im unbestückten Zustand besser durchschneiden läßt.

Beim Bestücken und Verlöten der Karte ist größte Sorgfalt geboten. Eventuelle Kurzschlüsse können nicht nur zu Fehlfunktionen der Interface-Karte führen, sondern auch den Rechner selbst beschädigen! Eine schlechte Lötstelle oder eine falsche Bestückung haben immer eine Fehlfunktion der Karte oder des gesamten Rechners zur Folge. Derjenige Benutzer der Interface-Karte, der noch keine oder nur wenig Erfahrung beim Aufbau von elektronischen Schaltungen auf Platinen hat, sollte vor dem Bestücken die Anweisung im Anhang (Kapitel 9.2) lesen. Bild 8.5 zeigt den Bestückungsplan der Platine. Bild 8.6 zeigt einen Musteraufbau der Platine als Fotografie. Die entsprechende Schaltung ist dem Bild 8.7 zu entnehmen.

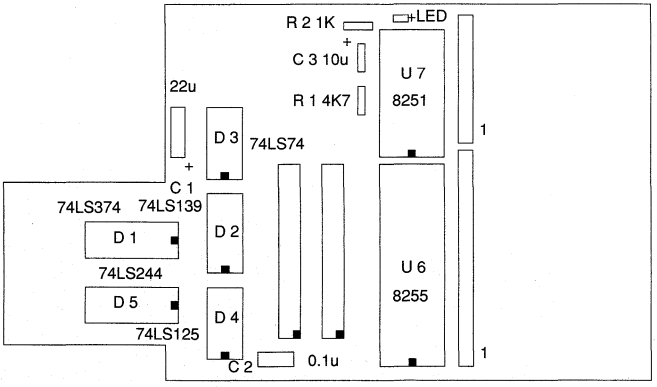


Bild 8.5: Bestückungsplan des Logikteils der Platine

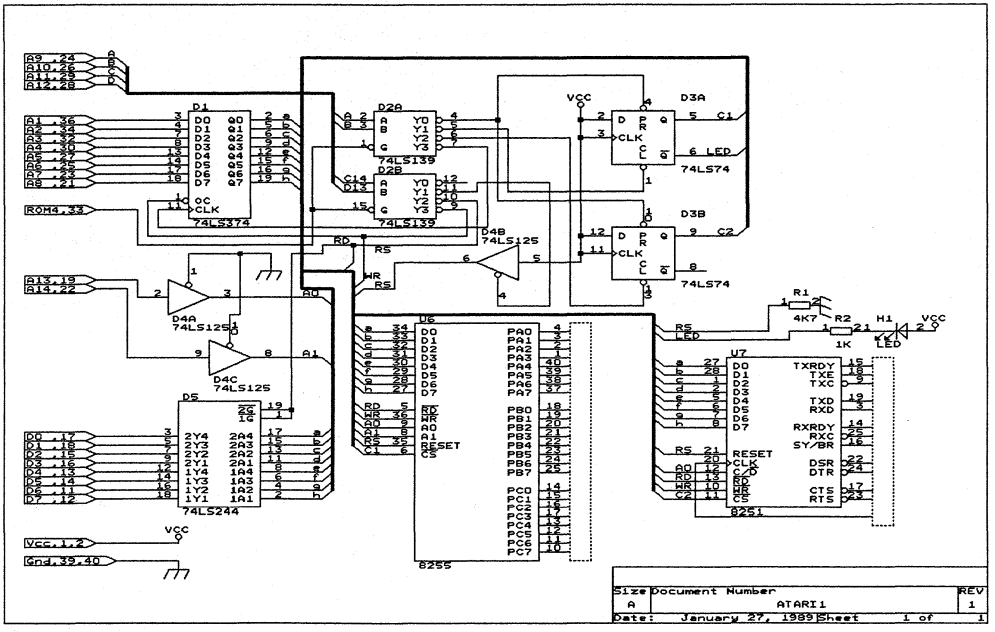


Bild 8.6: Die Schaltung der Interface-Platine (Logikteil)
(Einen vergrößerten Schaltplan finden Sie im Anhang S. 270)

Die Platine ist sowohl mit TTL-ICs als auch mit CMOS-ICs in allen beschriebenen Funktionen einsatzbereit. Man sollte jedoch eine Mischbestückung vermeiden. Eine Ausnahme bildet der IC SN74LS125, der z.Z. nur in einer TTL-Version verfügbar ist. Bevor man ir-

gendwelche Bauteile in die Platine einlötet, wird sie auf einwandfreie Ausführung geprüft. Man sollte sie auf Kurzschlüsse oder Unterbrechungen genau untersuchen und Fehler gegebenenfalls reklamieren. Mit einem Ohmmeter läßt sich zusätzlich auch ein fataler Kurzschluß zwischen dem Plus- und Minus-Pol der Platine sofort herausfinden.

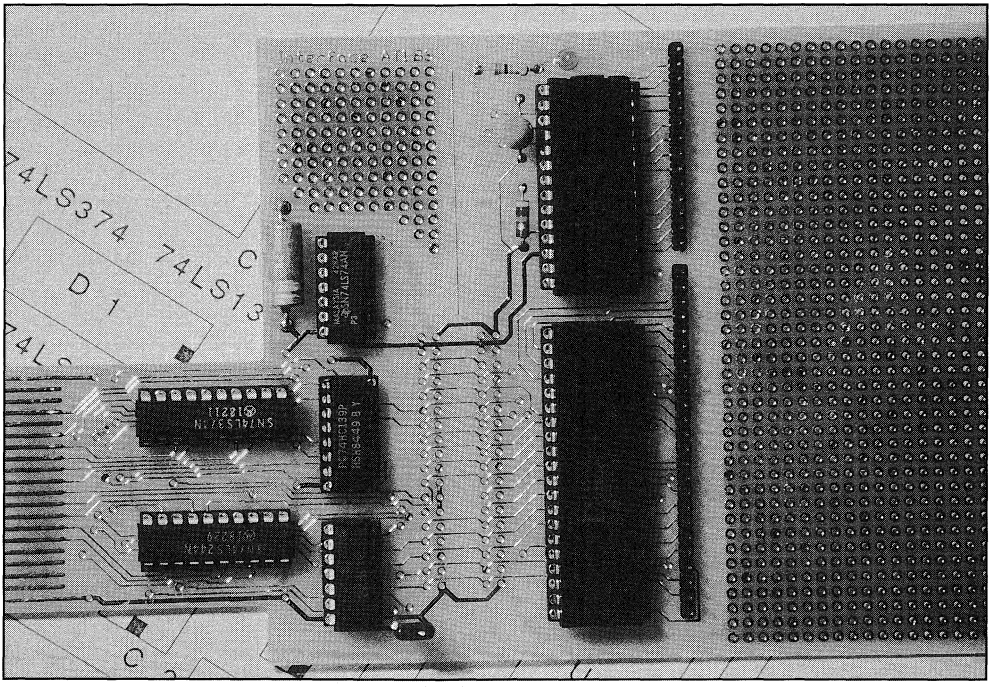


Bild 8.7: Musteraufbau der Platine mit bestücktem Logikteil

Zur Inbetriebnahme setzt man alle passiven Bauteile in die Karte ein und verlötet diese. Dazu gehören die IC-Sockel, die Widerstände, die Kondensatoren, die Leuchtdiode und gegebenenfalls die beiden Verbindungsstecker. Bei den Kondensatoren und der Leuchtdiode ist auf die richtige Polarität zu achten. Rote Leuchtdioden haben in der Regel am positiven Pol (+) einen längeren Drahtanschluß. Die Elektrolyt- und Tantalkondensatoren sind durch ihren Aufdruck am Pluspol gekennzeichnet. Die integrierten Schaltkreise werden noch nicht eingesetzt. Nun hat man Gelegenheit, alle Lötstellen nochmals zu kontrollieren und diese auf Güte oder auf eventuelle Kurzschlüsse zu untersuchen. Nach dem Abschalten des Rechners setzt man die Platine in den Atari-ROM-Port ein wie dies Bild 8.8 zeigt. Nach dem Einschalten des Rechners muß sich das Betriebssystem in der gewohnten Weise melden. Die Leuchtdiode darf nicht aufleuchten. Sollten beim Hochfahren des Systems irgendwelche Besonderheiten auftreten, so muß nach der Fehlerbehandlung in Kapitel 9.1 Teil A vorgegangen

werden. Sofern sich das Betriebsprogramm in der gewohnten Weise gemeldet hat, schaltet man den Rechner wieder ab und setzt die ICs D1 bis D5 ein. Sie erhalten auf der Platine folgende Position:

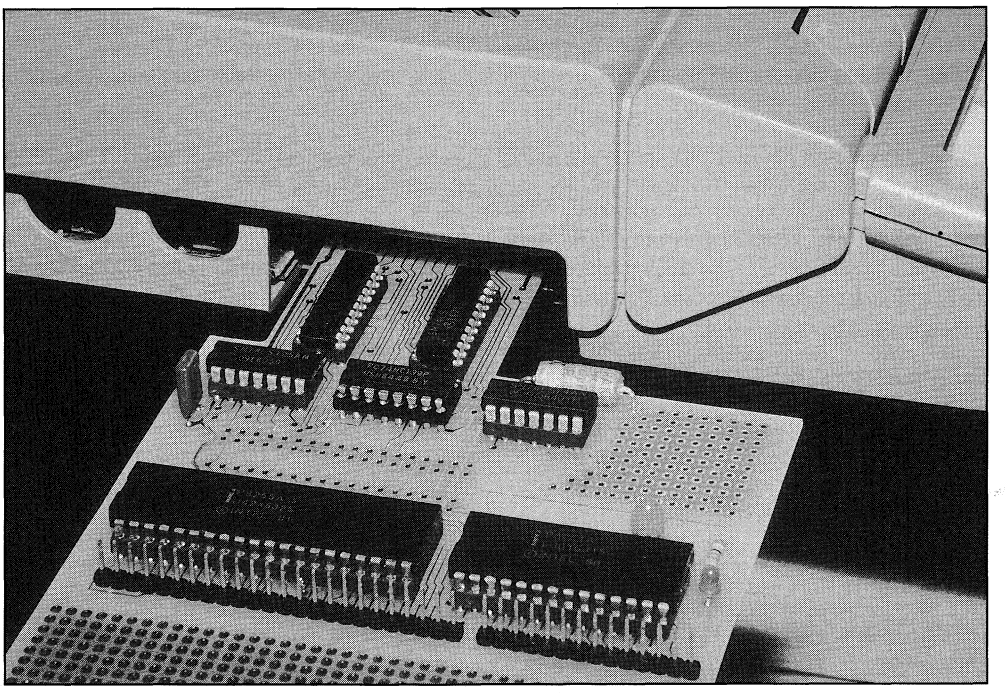


Bild 8.8: Die Interface-Platine im Atari ST

1 SN 74LS374	Latch	D1	1 SN 74LS125	Treiber	D4
1 SN 74LS139	Dekoder	D2	1 SN 74LS244	Treiber	D5
1 SN 74LS74	D-Flipflop	D3			

Alle ICs müssen an den richtigen Stellen der Platine Platz finden und ordnungsgemäß ausgerichtet sein. Die Markierung zeigt dabei stets nach rechts bzw. nach unten (Platine liegt wie im Bild 8.5). Die falsche Orientierung eines ICs kann zum Ausfall der Netzversorgung im Rechner führen. Nach dem erneuten Einschalten des Rechners sollte sich das Betriebssystem wieder einwandfrei melden. Die Leuchtdiode kann (vom Zufall abhängig) nun auch aufleuchten. Sollten an dieser Stelle Schwierigkeiten bei der Inbetriebnahme auftreten, so ist nach der Fehlerbehebung Teil B vorzugehen.

Zuletzt bestückt man die beiden Interfacebausteine in folgenden Positionen:

1 8255A	Par. Interf.	U6	8251A	Ser. Interf.	U7

Auch diese abschließende Arbeit der Bestückung wird erneut durch eine passive Funktionskontrolle am Rechner überprüft. Sollte sich dabei das Betriebsprogramm nicht mehr melden, oder eines der ICs sehr heiß werden, so ist nach der Fehlerbehandlung Teil C vorzugehen.

Bei gelungenem Aufbau untersucht man die Platine nochmals auf mögliche Fehler (schlechte Lötstellen oder unterlassene Lötungen) und beginnt dann mit dem aktiven Test der Platine. Dabei werden die Funktionseinheiten durch einfache Programme aktiviert und in ihrem Ablauf überprüft.

Die beiden ersten Programme betreiben die Leuchtdiode und geben damit bereits Aufschluß über einen wesentlichen Teil der Funktionsbereitschaft der Platine.

Sie sind in der Programmiersprache Basic geschrieben und dienen lediglich dazu, die Leuchtdiode auf der Platine anzusteuern. Dabei schaltet das Programm L_ON die Leuchtdiode ein, während das Programm L_OFF die Leuchtdiode ausschaltet. Die beiden Programme haben folgendes Aussehen:

Programm L_ON

```
a%=PEEK(16384000)
END
```

Programm L_OFF

```
a%=PEEK(16384512)
END
```

Die Programme können hintereinander zum Ablauf gebracht werden und sollten dann die entsprechende Funktion bei der Leuchtdiode hervorrufen. Ein fehlerhaftes Verhalten kann durch die Fehlerbehebung (Teil D) behoben werden.

Natürlich ergibt sich aus den beiden Kurzprogrammen für ein Ein- oder Ausschalten der Leuchtdiode sofort die sinnvolle Kombination eines kleinen Blinkprogramms, das im folgenden dargestellt ist:

Programm Blink

```
REPEAT
  a%=PEEK(16384000)
  FOR i=1 TO 2000
  NEXT i
  a%=PEEK(16384512)
  FOR k=1 TO 2000
  NEXT k
UNTIL INKEY$=CHR$(27)
END
```

Das Blinkprogramm kann man durch die Esc-Taste unterbrechen.

8.2 Funktionsweise der Interface-Platine

Sofern die ersten drei Interfaceprogramme (aus Kapitel 8.1) alle gewünschten Funktionen zur Zufriedenheit ausgeführt haben, bleibt vor der weiteren Bearbeitung nur noch die Frage zu klären, wie die Platine eigentlich funktioniert. Verblüffenderweise ermöglicht der Atari-ROM-Port nur das Einlesen von Daten. Jede Schreib-Funktion führt zu einer Systemunterbrechung und kann daher nicht genutzt werden. Das Blinken der Leuchtdiode ist aber doch offensichtlich eine Ausgabe-Funktion, die in irgendeiner Form durch eine Eingabe zustande kommt.

Das Geheimnis, das hinter dieser Pseudo-Schreib-Funktion steckt, besteht darin, daß man bei jeder Lese-Operation zuerst eine gewisse Adresse schreiben muß, auf die man zugreifen möchte. Genau diese Eigenschaft macht sich die Platine mit der entsprechenden Software zunutze. Sie verwendet einfach bestimmte Adressen als Daten und benutzt diesen Bereich zur Schreibfunktion. Bild 8.9 veranschaulicht die Wirkungsweise. Die Adreßleitungen A12, A11, A10 und A9 gelangen auf der Platine direkt an zwei getrennte Dekoder (ausgebildet als SN74LS139). Jede Lese-Funktion betreibt an den Ausgängen der beiden Dekoder die entsprechende Leitung. Dabei dient der erste Dekoder (links) zur Anwahl des Bausteins oder der LED auf der Platine, während der zweite Dekoder (rechts) die Funktion durchführt (z.B. Schreiben oder Lesen). Um eine zeitliche Sequenz zu generieren ist hinter dem linken Dekoder noch ein zweifaches D-Flipflop geschaltet, das die letzte Stellung des Dekoders nach der Lese-Funktion beibehält.

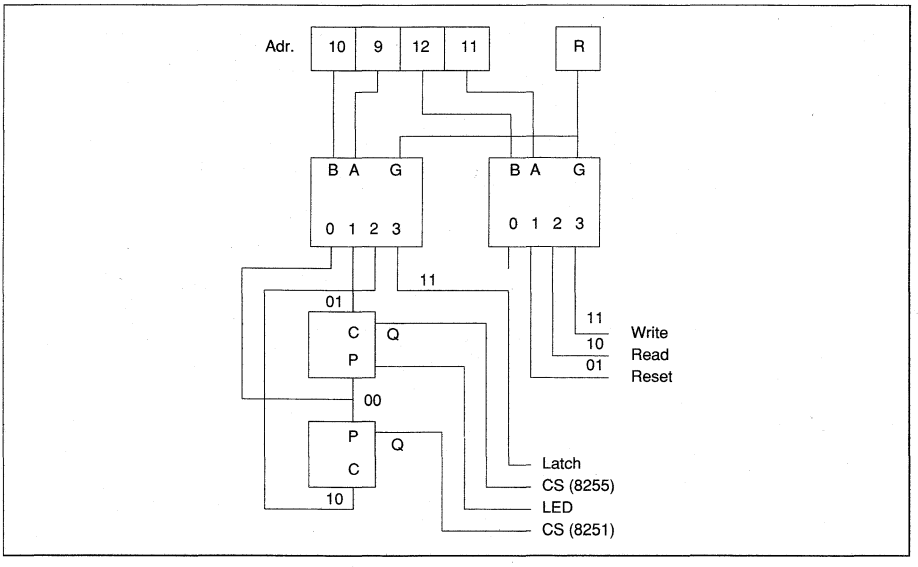


Bild 8.9: Blockschaltbild der Dekodierlogik auf der Interface-Platine

Die Lese-Funktion auf dem ROM-Port wird vom Atari automatisch durch eine Adreßanwahl auf der hexadezimalen Adresse FA0000-FAFFF durchgeführt (R-Signal). Ein Lesezugriff auf eine dieser Adressen (z.B. durch PEEK 16384000) steuert den linken Dekoder derart an, daß er die beiden nachgeschalteten D-Flipflops in den Preset-Zustand versetzt. Die Chip-Select-Signale (CS 8255 und CS 8251) nehmen damit den logischen Zustand "1" an. Die Leitung LED wird dagegen auf logisch "0" gesetzt. In diesem Fall muß die Leuchtdiode aufleuchten, da sie mit ihrem positiven Pol an +5 Volt liegt.

Entsprechend ergibt sich für eine Write-Funktion über den rechten Dekoder folgende Adressierung: Zur Dekodierung des Ausgangs 3 am Decoder (Write) müssen die beiden Eingänge A und B auf logisch "1" gebracht werden. Die Adreßleitungen A12 und A11 sind demnach auf "1"-Potential zu setzen. So führt der Lesebefehl der Adresse FA1800 hex (oder PEEK 16390144) zu einem "Write-Impuls".

Nach dem Schema von Bild 8.9 lassen sich damit alle Adressierungarten und Ansteuerprozeduren der Platine genau berechnen. Bild 8.10 zeigt die Verwendung der restlichen Adressen und deren Wirkungsweise innerhalb der Platine. So ist sofort erkennbar, daß die Adreßleitungen A1 bis A8 nur zur Speicherung in einem Latch dienen und damit als interner 8-Bit-Datenbus zu verstehen sind. Eine Dateninformation gelangt in die Platine, indem man eine geeignete Adresse anwählt und zusätzlich am Dekoder die Latch-Funktion wählt. Einmal abgespeicherte Daten bleiben später auf der Platine erhalten und können in weiteren Schritten den internen Bausteinen zur Verfügung gestellt werden. Die Adreßleitungen A13 und A14 sind direkt mit dem Baustein 8255 verbunden und ermöglichen die interne Anwahl im parallelen Port.

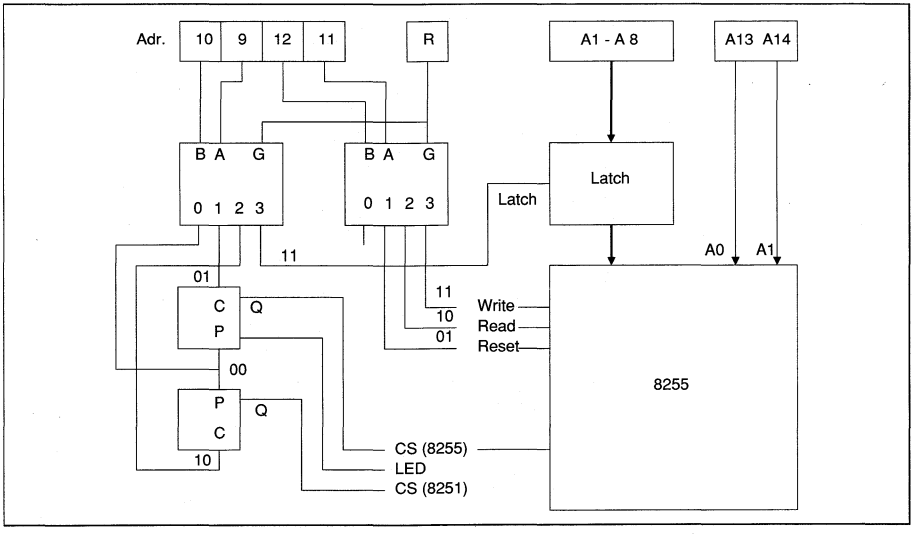


Bild 8.10: Dekodierlogik mit den Kontroll-Leitungen des 8255

Für den späteren Ausbau der Platine befindet sich am Decoder bereits eine Selektierlogik für den seriellen Baustein 8251.

Die ersten Applikationen beziehen sich jedoch auf den Baustein 8255. Seine Möglichkeiten bieten bereits zahlreiche Anwendungen, ohne daß ein weiterer Ausbau der Platine erforderlich ist. Dabei wird der Baustein in zwei unterschiedlichen Betriebsarten programmiert, welche die 24 Portleitungen in ihren Funktionen festlegen. Bild 8.11 zeigt den Baustein 8255 mit den Portleitungen A0 bis B7. Sie sind auf den unteren Interfaceadapter geführt, wobei die Nummer 1 (also A0) ganz unten liegt.

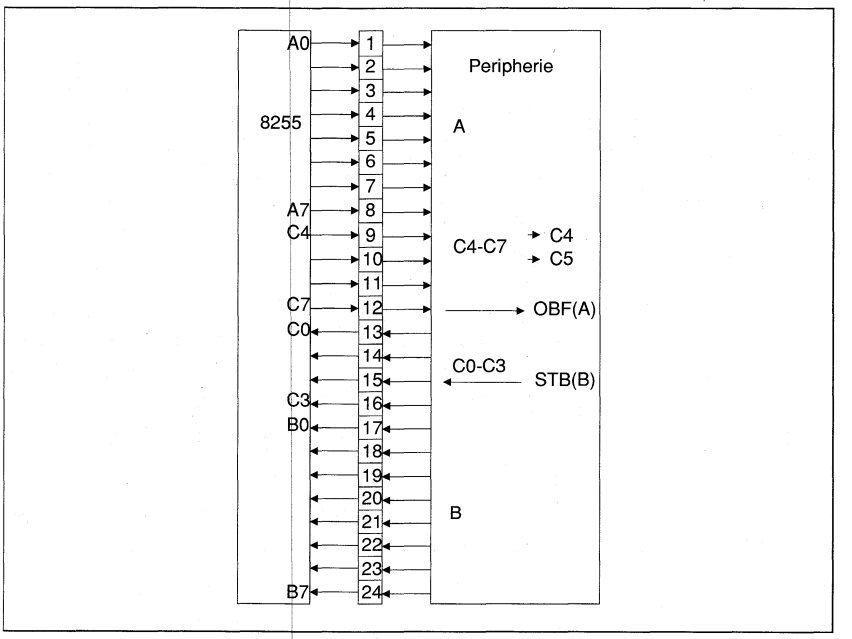


Bild 8.11: Alle Portleitungen des 8255 und deren Belegung auf der Platine

Beide Programmierarten benutzen die Ports von A als Ausgabe und die Ports von B zur Eingabe. Im ersten Fall (Programmierwort 83 hex) haben die Ports C0 bis C3 eine Eingabe- und die Ports C4 bis C7 eine Ausgabefunktion. Durch das zweite Programmierwort kann man den Baustein im Mode 1 betreiben (Programmierwort A6 hex). Durch diese Wahl gelingt eine Strobe-Verarbeitung für den Port B durch den Eingang C2, während ein Datenwort am Port A durch das OBF-Signal am Ausgang C7 signalisiert wird. Die Ports C4 und C5 sind dann als Ausgabe geschaltet. Ein Beispiel soll die richtige Adressierung vorstellen. Die Adreßtabelle hat folgendes Aussehen:

```
Adresse    A14 A13 A12 A11 A10 A9    A8-A1

Funktion

Schreiben          1   1
Lesen              1   0
Reset              0   1
Select (CS)                0   1
Latch                      1   1

           A1   A0                    Daten (des 8255)
```

Wie schon vorher beschrieben, ist die Basisadresse FA0000. Zu ihr werden danach alle notwendigen Adreßkombinationen addiert, die der Funktion und dem Datenwort entsprechen. Möchte man z.B. das Datum 83Hex in den Kontrollport des 8255 laden, so braucht man insgesamt drei Operationen. Bei der ersten wird der Wert 83Hex im Latch abgelegt. Die notwendige Bitkodierung hat dabei folgende Form:

```
0 0 0  0 1 1 1  0 0 0 0  0 1 1 0
```

Die ersten drei Nullen entfallen auf die Adressen A14 bis A12. Sie verbleiben auf 0, da keine Schreib- oder Lesefunktion gefordert wird. Die nächsten 4 Bit haben die Form 0 1 1 1 und entfallen auf den Adreßbereich A11 bis A8. Mit ihnen wird die Funktion "Latch" angesprochen. Zusätzlich folgt die Information 83Hex, bei der zu berücksichtigen ist, daß A0 stets auf 0 bleibt. Die Information 83Hex erscheint damit um eine Stelle nach links verschoben, so daß A7 in das Bit von A8 übergeht.

Nach der ersten Information muß nun eine Schreibfunktion auf den Kontrollport des 8255 erfolgen. Dazu wird die Schreib-Kodierung und das Selektiersignal (CS) angesprochen:

```
1 1 1  1 0 1 0  0 0 0 0  0 0 0 0
```

Abschließend schaltet man alle Signale wieder aus, so daß der Baustein im Ruhezustand verweilt (Kodierung 000 0000 0000 0000).

Die gesamte Sequenz hat damit im Bezug auf die Basisadresse folgendes Aussehen:

FA0706	entspricht	16385798	83Hex auf Latch
FA7A00	entspricht	16415232	WR,CS mit A0,A1=1
FA0000	entspricht	16384000	alles wieder aufheben

8.3 Einfache Anwendungen der Platine

Die beiden folgenden Programme sind erneut in der Sprache Basic geschrieben und geben detailliert einen Überblick, wie eine vollständige Programmierung mit einer einfachen Funktionsweise zustandekommt. Die Programme sind derart ausgelegt, daß jede einzelne Adreßleitung verarbeitet wird. Sie sind daher in der Ausführung etwas langsam, dennoch zeigen sie, wie eine sinnvolle Programmiersequenz erfolgen kann und sich die eigentlichen Adressen zusammensetzen.

Das erste Programm COUNT8B benutzt den Ausgabeport A als kontinuierlichen Ausgabezähler. Dabei erscheint am Port A0 eine feste Frequenz, die von der Laufzeit des Basic-Programms abhängt. A1 gibt die durch 2 geteilte und A2 die durch 4 geteilte Frequenz wieder. An A7 erscheint eine durch 128 geteilte Frequenz von A0.

Programm COUNT8B

```
REM Counter-Ausgabeprogramm
REM
REM Zuerst wird der 8255-Baustein programmiert
REM
REM Konfigurationsprogramm (8255)
REM Port A,C4-C7 Ausgabe  B,C0-C3 Eingabe
REM
REM A9-A14 Adressleitungen
REM C Kontrollwort
REM
REM CS (8255) = 1
a9=0
a10=0
a11=0
a12=0
a13=1
a14=1
c=131
a1=PEEK(16384000+16384*a14+8192*a13+4096*a12+2048*a11+1024*a10+512*a9+2*c)
REM
REM Reset (8255)
a11=1
a1=PEEK(16384000+16384*a14+8192*a13+4096*a12+2048*a11+1024*a10+512*a9+2*c)
REM
REM Reset entfernen, Daten im Latch speichern
a9=1
a10=1
```

```
a11=0
a1=PEEK(16384000+16384*a14+8192*a13+4096*a12+2048*a11+1024*a10+512*a9+2*c)
REM
REM CS (8255) = 0
a9=1
a10=0
a1=PEEK(16384000+16384*a14+8192*a13+4096*a12+2048*a11+1024*a10+512*a9+2*c)
REM
REM WR (8255) = 0
a11=1
a12=1
a1=PEEK(16384000+16384*a14+8192*a13+4096*a12+2048*a11+1024*a10+512*a9+2*c)
REM
REM WR, CS (8255) = 1
a9=0
a11=0
a12=0
a1=PEEK(16384000+16384*a14+8192*a13+4096*a12+2048*a11+1024*a10+512*a9+2*c)
REM
REM Nach der Programmierung erfolgt die Ausgabe auf den Port A
REM
c=0
a13=0
a14=0
REPEAT
  a9=1
  a10=1

a1=PEEK(16384000+16384*a14+8192*a13+4096*a12+2048*a11+1024*a10+512*a9+2*c)
  a10=0
  a11=1
  a12=1

a1=PEEK(16384000+16384*a14+8192*a13+4096*a12+2048*a11+1024*a10+512*a9+2*c)
  a1=PEEK(16384000)
  a11=0
  a12=0
  c=c+1
  IF c=256 THEN
    c=0
  ENDIF
UNTIL INKEY$=CHR$(27)
END
```

Das Programm läßt sich durch die Taste Esc unterbrechen. Die richtige Funktionsweise kann durch ein Oszilloskop oder an den Ausgängen A6 und A7 bereits durch ein Voltmeter oder durch eine Leuchtdiode überprüft werden (bitte den Maximalstrom von 2 mA nicht überschreiten!).

Das Programm INPUT8B ist ganz ähnlich aufgebaut wie das Zählprogramm COUNT8B. Es benutzt lediglich den Eingabeport B und stellt dessen Zustand als Zahl auf dem Bildschirm dar. Die ausgegebene Intergerzahl beinhaltet dabei in den letzten 8 Bit die Eingabeinformation. Sofern alle acht Leitungen logisch "0" sind muß die Zahl 65280 erscheinen (entspricht FF00 hex). Eine gemeinsame "1"-Funktion sollte zu der Darstellung 65535 führen (entspricht FFFF hex).

Die Eingänge lassen sich durch das Anlegen der entsprechenden Spannung von 0 oder 5 Volt prüfen. Bei der CMOS-Version des 8255 (82C55 oder D71055C) reicht schon ein Berühren der Interfaceleiste zur Spannungseingabe. Das Programm INPUT8B hat folgendes Aussehen:

Programm INPUT8B

```
REM Eingabeprogramm fuer 8 Bit
REM
REM Das Programm beginnt mit der Konfiguration
REM
REM Konfigurationsprogramm (8255)
REM Port A,C4-C7 Ausgabe  B,C0-C3 Eingabe
REM REM A9-A14 Adressleitungen
REM C Kontrollwort
REM
REM CS (8255) = 1
a9=0
a10=0
a11=0
a12=0
a13=1
a14=1
c=131
a1=PEEK(16384000+16384*a14+8192*a13+4096*a12+2048*a11+1024*a10+512*a9+2*c)
REM
REM Reset (8255)
a11=1
a1=PEEK(16384000+16384*a14+8192*a13+4096*a12+2048*a11+1024*a10+512*a9+2*c)
REM
REM Reset entfernen, Daten im Latch speichern
a9=1
```

```
a10=1
a11=0
a1=PEEK(16384000+16384*a14+8192*a13+4096*a12+2048*a11+1024*a10+512*a9+2*c)
REM
REM CS (8255) = 0
a9=1
a10=0
a1=PEEK(16384000+16384*a14+8192*a13+4096*a12+2048*a11+1024*a10+512*a9+2*c)
REM
REM WR (8255) = 0
a11=1
a12=1
a1=PEEK(16384000+16384*a14+8192*a13+4096*a12+2048*a11+1024*a10+512*a9+2*c)
REM
REM WR, CS (8255) = 1
a9=0
a11=0
a12=0
a1=PEEK(16384000+16384*a14+8192*a13+4096*a12+2048*a11+1024*a10+512*a9+2*c)
REM
REM Einlesefunktion
REM
REPEAT
  a9=1
  a10=0
  a11=0
  a12=1
  a13=1
  a14=0

a1=DPEEK(16384000+16384*a14+8192*a13+4096*a12+2048*a11+1024*a10+512*a9+2*c)
  PRINT a1
UNTIL INKEY$=CHR$(27)
END
```

Auch dieses Programm läßt sich durch Drücken der Esc-Taste beenden. Es bleibt dem Programmierer überlassen, dieses Programm zu vereinfachen oder die Ausgabe in der gewünschten Form zu verändern (z.B. die Ausgabe nur als Byte oder als binäres Wort).

Alle weiteren Applikationen in diesem Buch machen von der ausführlichen Adressierung keinen Gebrauch mehr. Da das Prinzip in allen folgenden Programmen genau übernommen wird, sollte an dieser Stelle lediglich die Adreßverwaltung im einzelnen erläutert werden.

Wenn der Baustein 8255 im Mode 1 betrieben wird, lassen sich Eingaben des Port B über eine Strobe-Leitung einlesen. Der Port C2 dient dabei als Übernahmesignal. Mit jeder negativen Taktflanke wird die Information am Port B eingelesen. Die Strobe-Funktion dient damit zur Synchronisation mit der externen Hardware, die gegebenenfalls Signale nur kurzzeitig zur Verfügung stellt.

Die Eingabe einer solchen Strobe-Information läßt sich am Status des Port C ablesen. Dabei ist das Bit D1 solange "0", bis ein Byte in den Port B eingelesen wurde. Ein Informationswechsel nach "1" signalisiert, daß eine gültige Information im Port B vorliegt. Nach dem Einlesen des Port B wird dieses Bit automatisch wieder auf "0" gesetzt. Das Programm STROBE verarbeitet diese getriggerte Eingabe des Port B. Mit dem Aufruf "b" läßt sich der Inhalt des Port B und mit dem Aufruf "c" der Inhalt des Statusregisters lesen. Jede negative Flanke an der Signalleitung C2 bewirkt ein Setzen des Bits D1 im Statusregister. Diese "1" bleibt solange erhalten, bis man den Port B durch die Eingabe "b" ausliest.

Programm STROBE

```
REM Getriggerte Eingabe auf Port B
REM
GOSUB prog_stb
REPEAT
  x$=INKEY$
  IF x$="b"
    GOSUB read_b
    PRINT "b= ";BIN$(b,8)
  ENDIF
  IF x$="c"
    GOSUB read_c
    PRINT "c= ";BIN$(c,8)
  ENDIF
UNTIL x$=CHR$(&H1B)
END
REM
PROCEDURE prog_stb
  x=PEEK(16385884)
  x=PEEK(16384512)
  x=PEEK(16415232)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE read_b
  b=DPEEK(16396800)
  x=PEEK(16384000)
```

```
RETURN
REM
PROCEDURE read_c
  c=DPEEK(16404992)
  x=PEEK(16384000)
RETURN
```

8.4 Anwendungen und Programmierung in Assembler und Pascal

Basic-Programme haben zuweilen den Nachteil, daß sie zu langsam ablaufen. Auch kompilierte Versionen reichen für schnellste Applikationen selten aus. Dieses Kapitel beschäftigt sich daher mit den Programmiersprachen Assembler und Pascal, die bei zeitkritischen Anwendungen vorteilhaft eingesetzt werden können. Die Programme sind daher auch extrem einfach gehalten und geben lediglich darüber Auskunft, wie eine richtige Adressierung erfolgen kann.

Die schnellste Betriebsart bei einer Interfaceanwendung stellt die Programmierung in Assembler dar. Diese Programmiersprache erfordert aber genaue Kenntnisse über den Prozessor und über seinen Befehlssatz. Da es für den Atari zahlreiche Assembler auf dem Markt gibt, sollte die richtige Programmierung und die Abfolge, wie ein Programm erstellt wird, im Prozessor- und Assembler-Handbuch nachgelesen werden. In jedem Fall wird mit Hilfe eines Editors ein Programm-File (Source) erstellt. Dieses gelangt danach in den Assembler, der die Syntax und die Zuordnung prüft. Bei richtiger Eingabe erstellt er eine relokative Datei, die durch den Linker zu ausführbarem Code gelangt. Der Linker vermag darüber hinaus auch andere Programme mit Assembler-Programmen zu verbinden. Das Programm FLASH stellt ein einfachstes Blinkprogramm dar (Funktion wie BLINK), bei dem die Adressierung und die Ansprache der Platine in der geeigneten Sequenz erfolgt.

Programm FLASH

```
                ; Leuchtdioden-Testprogramm (Einfachversion)
                ; Das folgende Programm steuert die Leuchtdiode
                ; im Blink-Modus an
                ;
begin:          move.w    (16384000),d4
                clr.w     d3
wait1:          addi.w    #1,d3
                cmpi.w    #50000,d3
                bne       wait1
                move.w    (16384512),d4
                clr.w     d3
```

```
wait2:          addi.w    #1,d3
                cmpi.w    #60000,d3
                bne       wait2
                bra       begin
```

Leider hat das Programm FLASH den Fehler, daß es sich nicht mehr unterbrechen läßt. Im Handbuch des Assemblers findet man daher zahlreiche systemeigene Routinen, die Unterbrechungen, Textausgaben oder Maus-Funktionen zulassen. Eine leichte Abänderung des Programms FLASH führt zum Programm LEDBLK, das sich durch den Systemaufruf "rawio" jetzt unterbrechen läßt. Darüber hinaus verwendet das Programm LEDBLK auch Makrodefinitionen, die bei der Assemblerprogrammierung erheblich zur Übersichtlichkeit beitragen. Die Ein- und Ausschaltzeiten der Leuchtdiode sind im Deklarationsteil als Konstanten in ihrer Dauer festgelegt.

Programm LEDBLK

```
                ; Leuchtdioden-Testprogramm (unterbrechbare Version)
                ; Das folgende Programm steuert die Leuchtdiode
                ; im Blink-Modus an
                ;
                ; Deklaration der Konstanten

led_on          equ 16384000
led_off         equ 16384512
                ;
pause1          equ 50000
pause2          equ 60000
                ;
                ; Definition der Makros

                .MACRO rawio
                move.w    #$ff,-(sp)
                move.w    #6,-(sp)
                trap      #1
                addq.w    #4,sp
                .ENDM

                .MACRO term
                clr.w     -(sp)
                trap      #1
                .ENDM

begin:          move.w    (led_on),d4
                clr.w     d3
wait1:          addi.w    #1,d3
                cmpi.w    #pause1,d3
```

```
                bne         wait1
                move.w      (led_off),d4
                clr.w       d3
wait2:          addi.w      #1,d3
                cmpi.w      #pause2,d3
                bne         wait2

                rawio
                cmpi.w      #0,d0
                beq         begin

                term
```

Neben zahlreichen anderen Programmiersprachen eignet sich die Hochsprache Pascal auch zur Ansteuerung der Interfacekarte. Pascal ist in der Ausführungszeit nur geringfügig langsamer als Assembler. Diese Sprache bietet jedoch den Vorteil, daß man erheblich einfacher und sicherer ein Programm erstellen kann als in Assembler oder Basic. Die eindeutige Zuordnung der Parameter und die einheitliche Struktur sprechen für sich.

Zur Verwendung von Interfaceroutinen bietet Pascal einige Möglichkeiten, die von der Art des Programms, der Ansprache der Interfacekarte und dem verwendeten Pascal-Compiler abhängen.

Eine sehr weit verbreitete Methode, Interfacetechnik in Pascal durchzuführen, besteht darin, Assemblerprogramme einzulagern. Dabei verstehen manche Compiler entweder direkt Assemblerbefehle oder sie erlauben das Hinzufügen von Hexadezimalcode, der bestimmte Assemblerbefehle ersetzt. Derartige Inline-Assembler-Befehle sind jedoch nur schwer handhabbar und tragen selten zur Übersichtlichkeit des Gesamtprogramms bei. Deshalb verwendet man erheblich häufiger Prozeduren oder Funktionen, die im Pascal-Programm als "External" deklariert werden. In einem Zusatzprogramm ruft das Pascal-Programm dann eine entsprechende Routine aus einer anderen Sprache auf, die z.B. als Assemblerprogramm Daten überträgt. Ein Beispiel eines solchen Pascal-Programms stellt das Programm MASTST dar.

Programm MASTST

```
PROGRAM Mastst (output);

VAR xwert:real;

PROCEDURE subtst (VAR x:real); External;

BEGIN

 xwert:= 131*2;
 subtst (xwert);

END.
```

In diesem kurzen Beispielprogramm wird eine externe Prozedur mit dem Namen "subtst" deklariert und im Hauptprogramm aufgerufen. Der Wert "xwert" dient dabei als Übergabevariable.

Manche Pascal-Compiler ermöglichen es auch, daß man über die Befehle "PEEK" und "POKE" direkt periphere Einheiten anspricht. Das Programm SHOW-B benutzt den "PEEK"-Befehl und ermöglicht das Einlesen des Ports B von der Interfacekarte.

Programm SHOW-B:

```
PROGRAM show_b (INPUT,OUTPUT);

(* Das Programm liest den Datenport B      *)
(* des 8255 ein und stellt das Ergebnis dar *)

VAR x:INTEGER;

PROCEDURE prog_io;
  BEGIN
    x:= peek (0fa0706H);            (* Latch, Prg-Wort auf 83H *)
    x:= peek (0fa0200H);            (* C1=0 *)
    x:= peek (0fa7a00H);            (* WR=0, A0 A1=1 *)
    x:= peek (0fa0000H)             (* WR, C1=1 *)
  END; (* prog_io *)

PROCEDURE get_b;
  BEGIN
     x:= peek (0fa0000H);    (* C1, WR, RD=1 *)
     x:= peek (0fa3200H);    (* RD=0, A13=1 *)
  END; (* get_b *)

BEGIN (* show_b *)
  prog_io;
     REPEAT
      get_b;
      WRITELN (x);
     UNTIL x=0
END.
```

Das Programm arbeitet ähnlich wie das Basic-Programm "INPUT8B", wobei es durch eine Eingabe von acht Nullen am Eingaberegister des Ports B abgebrochen wird.

Als Beispiel soll auch das Programm PUT-A dienen, indem der Ausgabeport A durch den hexadezimalen Wert 55 auf eine alternierende Folge gesetzt wird.

Programm PUT-A:

```
PROGRAM put_a (OUTPUT);

(* Das Programm programmiert den 8255 und setzt  *)
(* danach den Ausgabe-Port A auf den Hex-Wert 55 *)

VAR x:INTEGER;

PROCEDURE prg_io;

   BEGIN
     x:= peek (0fa0706H);        (* Latch und Prg-Wort auf 83H *)
     x:= peek (0fa0200H);        (* C1=0 *)
     x:= peek (0fa7a00H);        (* WR=0, A0 A1=1 *)
     x:= peek (0fa0000H)         (* WR=1, C1=1 *)
   END; (* prg_io *)

PROCEDURE set_a;

   BEGIN
     x:= peek (0fa06aaH);        (* Latch und Datenbyte auf 55H *)
     x:= peek (0fa1a00H);        (* C1=0, WR=0, A0 A1=0 *)
     x:= peek (0fa0000H)
   END; (* set_a *)

BEGIN (* put_a *)

   prg_io;
   set_a

END.
```

8.5 Betrieb eines Logikanalysators mit der Interfacekarte

Logikanalysatoren sind ausgezeichnete Hilfsmittel bei der Fehlersuche und beim Test digitaler Schaltungen. Sie sind in der Lage, mehrere digitale Signale parallel aufzunehmen und deren zeitlichen Verlauf zu protokollieren. Anhand der zeitsynchronen Darstellung kann man die Funktionsweise einer Schaltung in allen Details studieren.

Ein besonderes Hilfsmittel innerhalb des Logikanalysators bildet ein Ereignistrigger, der eine spezielle Konstellation der digitalen Signale aufsucht. Man kann mit der zeitlichen Darstellung der logischen Pegel alle digitalen Funktionen untersuchen, die aus der Vergangenheit zum Triggerereignis geführt haben. Dabei stellt der Logikanalysator sowohl die Vorgeschichte als auch den weiteren Ablauf dar. Eine Fehlersuche mittels des Ereignistriggers

erfolgt daher besonders einfach, da man nach Abschluß der Signalaufnahme sowohl alle Pegel erfaßt hat, die zur Triggerung geführt haben und darüber hinaus auch noch deren zukünftige Wirkung untersuchen kann.

Der Logikanalysator muß für eine derartige Verarbeitungsweise kontinuierlich Signale von der Peripherie aufnehmen und diese zyklisch in einem Datenspeicher ablegen. Nur so ist er in der Lage, auch nach dem Auftreten der Triggerbedingung die Vorgeschichte darzustellen.

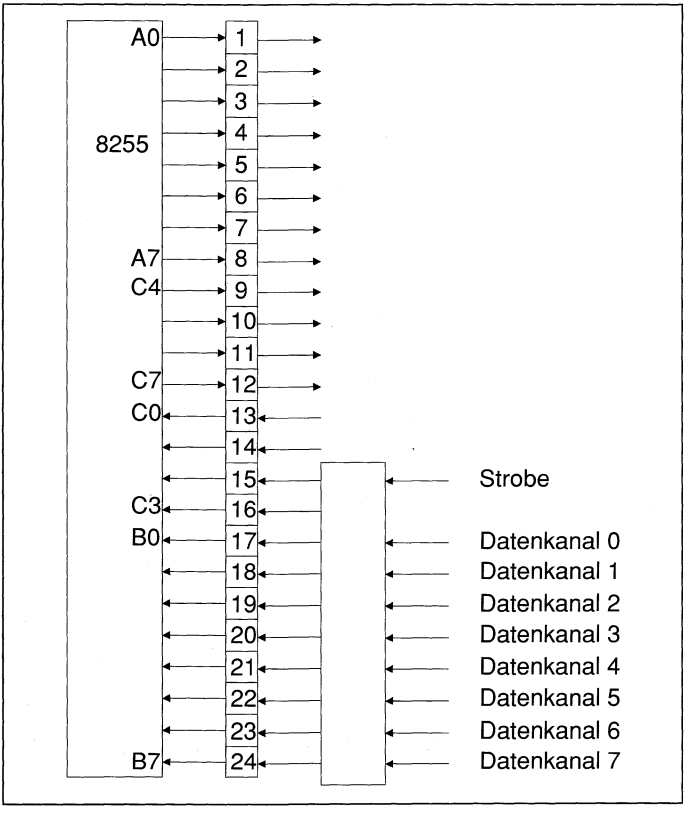


Bild 8.12: Beschaltung des 8255 für den Logikanalysator

Mit der Interfaceplatine läßt sich bereits ohne den weiteren Ausbau ein derartiger Logikanalysator realisieren. Er ist in der Lage, acht digitale Signale entweder zeitsynchron oder durch externe Triggerung abzuspeichern. Die vorhandene Speichertiefe des zyklischen Speichers beträgt 68 Byte. Dabei werden 16 Byte zur Ablage der Vorgeschichte und 52 Byte zur Speicherung der Folgeentwicklung verwendet. Bei der zeitsynchronen Ablage der Daten kann die Taktrate von 0,1 bis 10 Sekunden erfolgen. Eine Zeitangabe von 0 Sekunden führt auto-

matisch zur getriggerten Eingabe über die Strobe-Leitung, so daß jeweils eine Eingangsinformation mit der negativen Flanke des Strobe-Eingangs abgelegt wird.

Der Ablauf des Logikanalysators wird durch die Eingabe der Zeit- und Triggerbedingungen gestartet (Funktion: T). Die Eingabe "Q" beendet das Basic-Programm. Mit "P" kann man das gesamte Bild als Hardcopy ausdrucken lassen. Die "+"-Taste erlaubt eine Zoom-Funktion in vier Stufen. Mit den Pfeil-Tasten läßt sich der Cursor im Abbildungsfeld bewegen. Im rechten Zeit- und Triggerbedingungsfenster erhält man eine Übersicht der gewählten Werte und der momentanen Cursorposition. Sobald der Dateneingabeablauf gestartet wurde, beginnt der Logikanalysator mit der zyklischen Speicherung. Nach dem Erscheinen der Triggerfunktion wartet er noch 52 Byte ab und stellt danach den gesamten Ablauf grafisch dar. Der Cursor wird dabei auf die Triggerbedingung gesetzt. Da bei manchen Anwendungen vergeblich auf die Triggerbedingung gewartet wird, läßt sich das Programm auch durch die Esc-Taste unterbrechen. In diesem Fall hat man nur noch auf das folgende Byte zu warten, bevor der Ablauf endet. Der Cursor verbleibt in diesem Fall an der ersten Stelle des Eingabebytes.

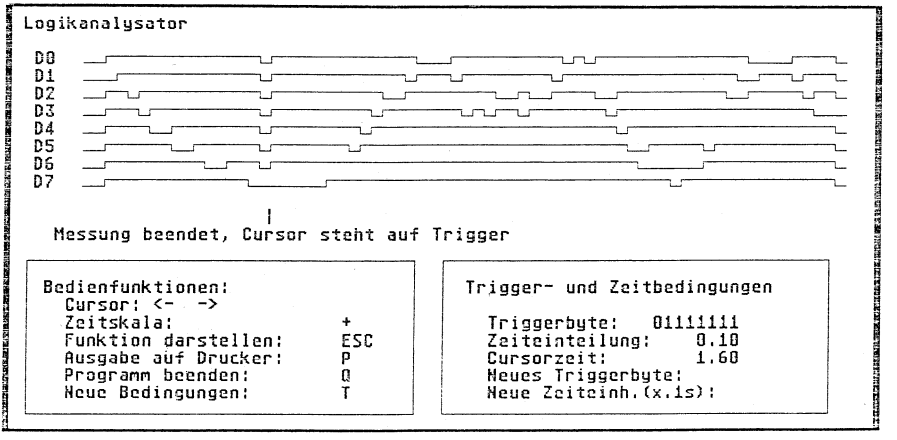


Bild 8.13: Hardcopy einer Logik-Analyse

Die notwendige Interface-Beschaltung des Bausteins 8255 wird im Bild 8.12 wiedergegeben. Wie schon in früheren Anwendungen dient der Port B des Bausteins als Eingabedatenkanal. Eine eventuelle Triggerung kann über den Port C2 (Strobe) erfolgen. Bei der Verwendung eines Interfacekabels sollte man ein Flachkabel verwenden, bei dem jede zweite Ader auf Masse gelegt wird. Mit einem derartigen Kabel kann man auch Schaltungen testen, die bis zu einem Meter von der Interface-Platine entfernt sind. Pegel unterhalb von 0 oder oberhalb von 5 Volt können zur Zerstörung des Bausteins 8255 oder gar zur Beschädigung des Atari ST führen. Bild 8.13 zeigt den Ausdruck eines typischen Bildes, wie es durch den Logikanalysator erstellt wurde. Das zugehörige Programm hat folgende Form:

Programm LOGIK_A

```
REM ********************************************
REM *      Demonstrationsprogramm eines          *
REM *          Logikanalysators                  *
REM ********************************************
REM
REM ********************************************
REM *            Hauptprogramm                   *
REM ********************************************
REM
DIM a|(68)
DIM h|(68)
m=0
t=0.1
c=0
z=1
y=0
GOSUB screen
REPEAT
  x$=INKEY$
  v$=RIGHT$(x$)
  IF v$=CHR$(&H4B)
    GOSUB left_cursor
  ENDIF
  IF v$=CHR$(&H4D)
    GOSUB right_cursor
  ENDIF
  IF x$="+"
    GOSUB zoom
  ENDIF
  IF x$="p"
    HARDCOPY
  ENDIF
  IF x$="t"
    GOSUB new_param
  ENDIF
UNTIL x$="q"
END
REM ***********************************************
REM ***           Unterprogramme                ***
REM ***********************************************
REM
```

```
PROCEDURE prog_io
  x=PEEK(16385798)
  x=PEEK(16384512)
  x=PEEK(16415232)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE prog_stb
  x=PEEK(16385884)
  x=PEEK(16384512)
  x=PEEK(16415232)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE read_b
  b=DPEEK(16396800)   b=b-65280
  x=PEEK(16384000)
RETURN
REM
PROCEDURE a_fenster
  CLS
  DEFLINE 1,3,0,0
  BOX 5,5,635,395
  PRINT AT(2,2);" Logikanalysator"
RETURN
REM
PROCEDURE info_fenster
  DEFLINE 1,1,0,0
  BOX 20,240,300,380
  PRINT AT(4,17);" Bedienfunktionen: "
  PRINT AT(6,18);" Cursor:   -"
  PRINT AT(6,19);" Zeitskala:                +"
  PRINT AT(6,20);" Funktion darstellen:      ESC"
  PRINT AT(6,21);" Ausgabe auf Drucker:      P"
  PRINT AT(6,22);" Programm beenden:         Q"
  PRINT AT(6,23);" Neue Bedingungen:         T"
RETURN
REM
PROCEDURE trig_fenster(VAR m,t,c)
  DEFLINE 1,1,0,0
  BOX 320,240,600,380
  PRINT AT(42,17);" Trigger- und Zeitbedingungen"
```

```
  PRINT AT(44,19);" Triggerbyte:  ",BIN$(m,8)
  PRINT AT(44,20);" Zeiteinteilung:"
  PRINT AT(44,21);" Cursorzeit:"
  PRINT AT(44,22);" Neues Triggerbyte:"
  PRINT AT(44,23);" Neue Zeiteinh.(x.1s):"
  IF t0
    PRINT AT(62,20);USING "###.##",t
    PRINT AT(62,21);USING "###.##",c*t
  ELSE
    PRINT AT(62,20);" getaktet"
    PRINT AT(62,21);c;" .Takt  "
  ENDIF
RETURN
REM
PROCEDURE bezeichnung
  PRINT AT(4,4);"D0"
  PRINT AT(4,5);"D1"
  PRINT AT(4,6);"D2"
  PRINT AT(4,7);"D3"
  PRINT AT(4,8);"D4"
  PRINT AT(4,9);"D5"
  PRINT AT(4,10);"D6"
  PRINT AT(4,11);"D7"
RETURN
REM
PROCEDURE bit_line
  n=0
  q=60
  x=ODD(a|(p)/s)
  PLOT q,o+6*x
  DRAW  TO q+8*z,o+6*x
  REPEAT
    x=ODD(a|(p+n)/s)
    DRAW  TO q+8*z,o+6*x
    DRAW  TO q+16*z,o+6*x
    n=n+1
    q=q+8*z
  UNTIL q>600
RETURN
REM
PROCEDURE display
  DEFLINE 1,1,0,0
```

```
  p=0
  o=60
  s=1
  i=1
  REPEAT
    GOSUB bit_line
    o=o+16
    s=s*2
  UNTIL s128
RETURN
REM
PROCEDURE set_cursor
  PRINT AT(c*z+9,13);CHR$(124)
  IF t0
    PRINT AT(62,21);USING "###.##",c*t
  ELSE
    PRINT AT(62,21);c;" .Takt  "
  ENDIF
RETURN
REM
PROCEDURE cl_cursor
  PRINT AT(c*z+9,13);CHR$(32)
RETURN
REM
PROCEDURE left_cursor
  IF c>0
    GOSUB cl_cursor
    c=c-1
    GOSUB set_cursor
  ENDIF
RETURN
REM
PROCEDURE right_cursor
  IF c*z<68
    GOSUB cl_cursor
    c=c+1
    GOSUB set_cursor
  ENDIF
RETURN
REM
PROCEDURE screen
  GOSUB a_fenster
```

```
  GOSUB bezeichnung
  GOSUB info_fenster
  GOSUB trig_fenster(m,t,c)
  GOSUB set_cursor
  GOSUB display
RETURN
REM
PROCEDURE zoom
  IF z<3
    z=z+1
  ELSE     z=1
  ENDIF
  GOSUB screen
RETURN
REM
PROCEDURE new_param
  LOCATE 22,66
  INPUT m
  IF m<0
    GOSUB error
  ENDIF
  IF m255
    GOSUB error
  ENDIF
  m=INT(m)
  LOCATE 23,66
  INPUT t
  IF t<0
    GOSUB error
  ENDIF
  IF t>100
    GOSUB error
  ENDIF
  t=0.1*(INT(t))
  GOSUB get_data
RETURN
REM
PROCEDURE error
  CLS
  PRINT AT(4,4);" Fehlerhafte Eingabe oder falscher Bereich!"
  FOR i=1 TO 30000
  NEXT i
```

```
  t=0.1
  m=0
  GOSUB screen
RETURN
REM
PROCEDURE get_data
  PRINT AT(6,14);"Messung aktiv, Unterbrechung mit ESC"
  ARRAYFILL a(),0
  ARRAYFILL h(),0
  IF t=0
    GOSUB prog_stb
  ELSE
    GOSUB prog_io
  ENDIF
  q=16
  u=0
  l=0
  d=0
  e=0
  w=0
  REPEAT
    GOSUB read_b
    a|(w)=b
    IF b=m AND l=1
      REPEAT
        GOSUB read_b
        h|(q)=b
        q=q+1
        GOSUB break         UNTIL q=68
      d=1
    ENDIF
    IF w>=16
      l=1
    ENDIF
    w=w+1
    IF w=68
      w=0
    ENDIF
    GOSUB break
  UNTIL d=1 OR e=1
  q=15
  IF d=1
```

```
    REPEAT
      n=a|(w)
      h|(q)=n
      q=q-1
      IF w=0
        w=68
      ENDIF
      w=w-1
    UNTIL q=0
    q=0
    w=0
    REPEAT
      n=h|(q)
      a|(w)=n
      q=q+1
      w=w+1
    UNTIL q=67
  ENDIF
  GOSUB screen
  IF d=1
    PRINT AT(6,14);"Messung beendet, Cursor steht auf Trigger"
    GOSUB cl_cursor
    c=16
    GOSUB set_cursor
  ENDIF
  IF e=1
    PRINT AT(6,14);"Messung unterbrochen"
  ENDIF
RETURN
REM
PROCEDURE break
  x$=INKEY$
  IF x$=CHR$(&H1B)
    e=1
  ELSE
    e=0
  ENDIF
  IF t=0
    REPEAT
      r=DPEEK(16404992)
      x=PEEK(16384000)
      r=r/2
```

```
    UNTIL ODD(r)
  ELSE
    PAUSE t*50
  ENDIF
RETURN
```

Wie alle Programme in diesem Buch ist auch das Logikanalysatorprogramm nur als Demonstrationsprogramm gedacht. Dennoch enthält es bereits zahlreiche Funktionen, die eine umfangreiche Arbeitsweise erlauben. Es bleibt jedem Anwender überlassen, gegebenenfalls Veränderungen oder spezielle Anpassungen an seine Wünsche durchzuführen.

8.6 Druckerinterface für parallele Drucker

Obwohl der Atari ST bereits über ein paralleles Interface zum Anschluß von Druckern nach der Centronics-Norm verfügt, ist es doch interessant, ein eigenes Interface aufzubauen. Zur Realisierung eines Druckerinterfaces muß man lediglich die Leitungen des Port A und einige Leitungen des Port C mit einer Druckerbuchse verbinden. Bild 8.14 zeigt die Beschaltung.

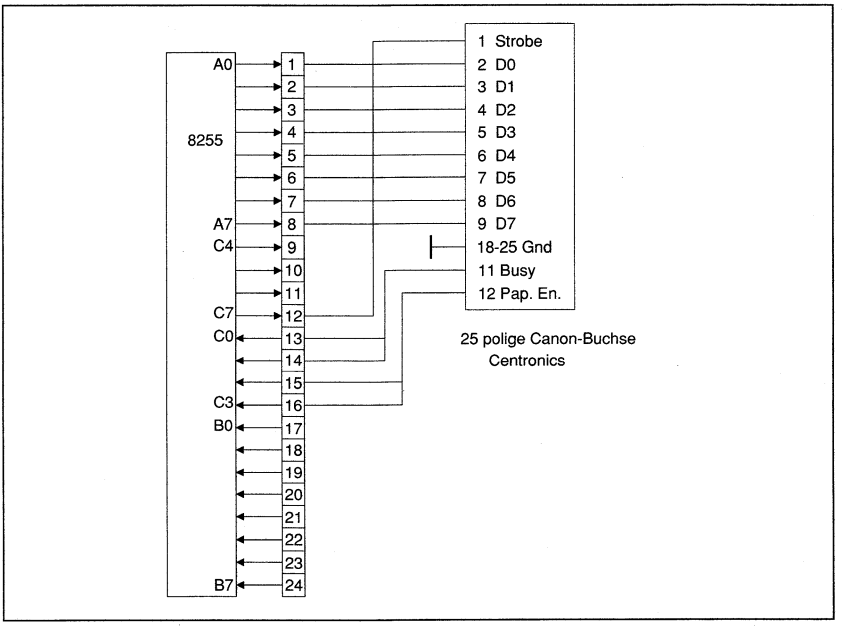


Bild 8.14: Anschlußbelegung des Drucker-Interface

Das folgende Basic-Programm bewirkt, daß alle ASCII-Zeichen vom "Space" bis zum "Z" mit einem "CR" und "LF" hintereinander ausgedruckt werden.

Programm DRUCK

```
REM ***********************************************
REM *      Drucker-Demonstrationsprogramm          *
REM ***********************************************
REM
GOSUB prog_io
c=255
GOSUB out_c
a=32
y=0
REPEAT
  GOSUB print_out
  a=a+1
UNTIL a=91 OR y<>0
IF y=1
  PRINT " Drucker ist nicht bereit!"
ENDIF
IF y=2
  PRINT " Drucker ist ohne Papier!"
ENDIF
IF y=0
  a=13
  GOSUB print_out
  a=10
  GOSUB print_out
ENDIF
END
REM
PROCEDURE prog_io
  x=PEEK(16385798)
  x=PEEK(16384512)
  x=PEEK(16415232)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE out_a
```

```
  x=PEEK(16385536+2*a)
  x=PEEK(16390656)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE out_c
  x=PEEK(16385536+2*c)
  x=PEEK(16407040)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE read_c
  c=DPEEK(16404992)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE print_out
  GOSUB read_c
  IF c=65523
    y=1
  ENDIF
  IF c=65535
    y=2
  ENDIF
  GOSUB out_a
  c=127
  GOSUB out_c
  c=255
  GOSUB out_c
RETURN
```

Im Gegensatz zum normalen Interface des Atari ST teilt diese Applikation dem Benutzer auch mit, ob der Drucker bereit ist und ob es ihm eventuell an Papier mangelt. Für eigene Anwendungen stellt das Programm DRUCK auch viele Unterprogramme (PROCEDURE) dar, die sich universell verwenden lassen. Zu diesen Prozeduren zählen die Unterprogramme prog_io, out_a, out_c und read_c. Die Namen dieser Prozeduren sprechen für sich, und der Benutzer tut gut daran, bei allen selbstgeschriebenen Programmen diese oder ähnliche Programmteile zu verwenden.

8.7 Weiterer Ausbau der Interface-Karte

Das freie Bestückungsfeld der Platine erlaubt den Aufbau eigener Applikationen unter Verwendung der bereits benutzten Bausteine. Die folgenden Kapitel dieses Buches beschäftigen sich mit dem individuellen Ausbau und geben zahlreiche Anleitungen, wie man zu speziellen Interfaceschaltungen gelangt. Alle Hardwareschaltungen lassen sich nacheinander auf der Platine unterbringen, ohne daß frühere Aufbauten wieder entfernt werden müßten. Neben den Beispielen hat man auch noch hinreichend Platz, zusätzliche Schaltungen auf der Platine zu integrieren.

Der weitere Ausbau erfolgt in zwei Schritten, wobei zuerst eine Schaltungserweiterung am 8255, danach eine Inbetriebnahme des seriellen Bausteins 8251 erfolgt.

In diesem Kapitel wird der parallele Baustein 8255 durch die aktiven Elemente eines Digital/Analog-Wandlers und eines Operationsverstärkers erweitert. Später erfolgt der Ausbau im Umfeld des seriellen Bausteins 8251. Die nun folgenden Applikationen sind nur dann sinnvoll, wenn die Anwendungsbeispiele der letzten Kapitel erfolgreich waren. Eventuelle fehlerhafte Aufbauten lassen sich nicht durch eine Weiterbestückung korrigieren.

Vor der Bestückung muß man sich die Bausteine ZN 426 (DAC) und LM 324 (OPA) besorgen. Einige passive Elemente sind ebenfalls notwendig. Bild 8.15 zeigt das Schaltbild des weiteren Ausbaus. Für die ersten Applikationen benötigt man zwar nur den Baustein ZN 426 und zwei der vier im LM 324 befindlichen Operationsverstärker. Dennoch sollte man bereits jetzt alle drei Operationsverstärker des Bausteins verdrahten, so daß man für spätere Anwendungen gerüstet ist. Bild 8.16 zeigt eine mögliche Verteilung dieser Komponenten auf der Platine.

Zur Bestückung sollte man erneut auf IC-Sockel zurückgreifen. Die fehlenden Leiterbahnen kann man durch Fädeldraht oder durch dünne Leitungen ersetzen. Wie bereits beim ersten Aufbau der Platine sollte man größte Sorgfalt walten lassen und insbesondere jetzt auf fehlerhafte Lötungen oder Kurzschlüsse achten. Ein abschließender Durchgangs- und Kurzschlußtest deckt noch vor der Inbetriebnahme mögliche Fehler auf.

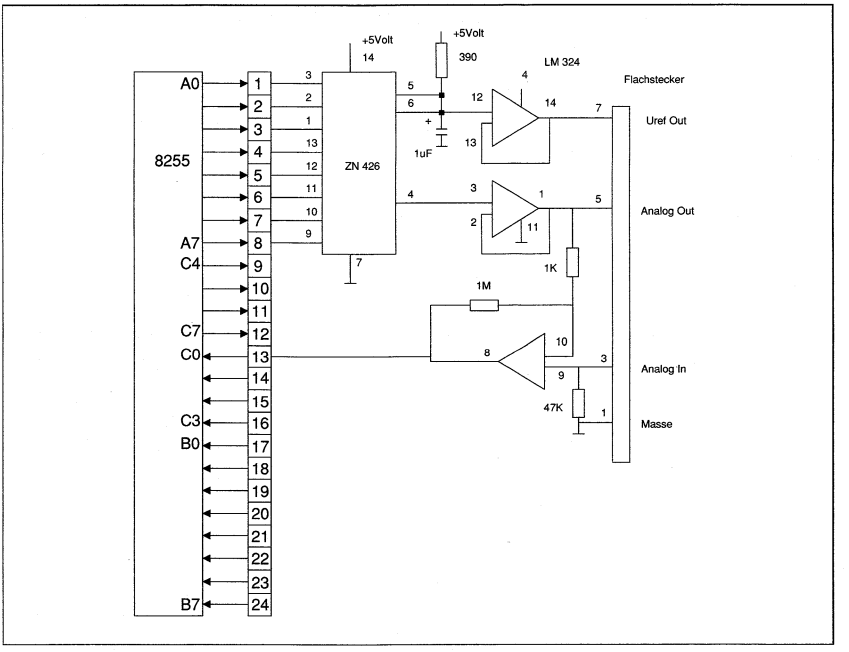


Bild 8.15: Schaltplan zur analogen Datenverarbeitung

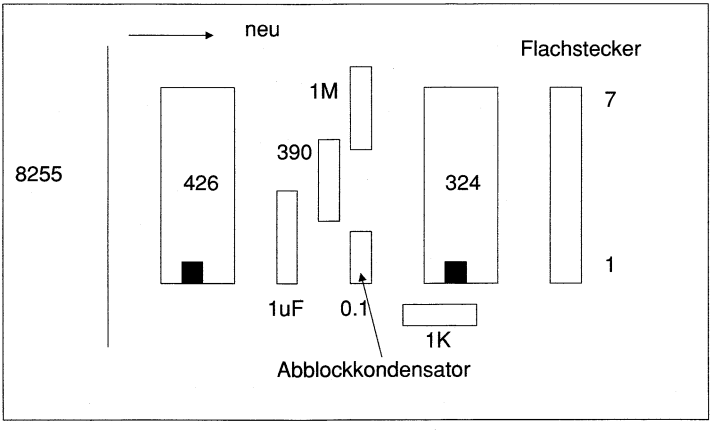


Bild 8.16: Bestückungsplan der Schaltung zur analogen Datenverarbeitung

8.8 Analoge Signalausgabe

Der Digital/Analog-Wandler ZN 426 ist ein Baustein, der 8 Bit breite Eingangsdaten in ein analoges Signal verwandelt. Er benötigt nur eine einzige Betriebsspannung und stellt zusätzlich eine stabile Referenzspannung von 2,5 Volt zur Verfügung.

Die Referenzspannung läßt sich am Ausgang des Operationsverstärkers (Uref Out) gegen Masse abgreifen. Die analoge Ausgangsspannung erhält man am Pin "Analog Out". Da der LM 324 ebenfalls ohne negative Spannung auskommt, sind keine zusätzlichen Versorgungen notwendig.

Das Basic-Programm ANAOUT ermöglicht die erste Ausgabe einer analogen Spannung, bei der man die Spannungshöhe vorwählen kann.

Programm ANAOUT

```
REM ***********************************************
REM ***            Analoge Ausgabe              ***
REM ***********************************************
REM
GOSUB prog_io
REPEAT
  INPUT "Ausgangsspannung (in mV):  ",a
  a=a*0.102
  a=INT(a)
  IF a<256
    GOSUB out_a
  ENDIF
UNTIL a255
END
REM
PROCEDURE prog_io
  x=PEEK(16385798)
  x=PEEK(16384512)
  x=PEEK(16415232)
  x=PEEK(16384000)
RETURN REM
PROCEDURE out_a
  x=PEEK(16385536+2*a)
  x=PEEK(16390656)
  x=PEEK(16384000)
RETURN
```

Der analoge Ausgang erlaubt selbstverständlich auch die Ausgabe von Signalmustern, wie sie bei Funktionsgeneratoren möglich sind. Das Programm DREIECK erzeugt eine Dreiecksfunktion mit einem Spannungshub von 2,5 Volt.

Programm DREIECK

```
REM ***********************************************
REM ***     Ausgabe einer Dreiecksfunktion      ***
REM ***********************************************
REM
GOSUB prog_io
a=0
REPEAT
  REPEAT
    GOSUB out_a
    a=a+1
  UNTIL a=256
  REPEAT
    a=a-1
    GOSUB out_a
  UNTIL a=0
  x$=INKEY$
UNTIL x$=CHR$(&H1B)
END
REM
PROCEDURE prog_io
  x=PEEK(16385798)
  x=PEEK(16384512)
  x=PEEK(16415232)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE out_a
  x=PEEK(16385536+2*a)
  x=PEEK(16390656)
  x=PEEK(16384000)
RETURN
```

Die Programmierung eines Funktionsgenerators in Basic stößt natürlich sehr schnell an die Grenze der Verarbeitungsgeschwindigkeit, wenn höhere Frequenzen gefordert sind. Eine Abhilfe kann hier die Assembler-Programmierung bringen, die zumindest den Vorstoß in den Kiloherzbereich zuläßt. Das Programm RAMPE ist ein Beispiel einer derartigen Assembler-Programmierung, bei der eine frei programmierbare Kurvenform am analogen

Ausgang erscheint. Bild 8.17 zeigt den Kurvenverlauf der Ausgangsspannung als Oszillogramm.

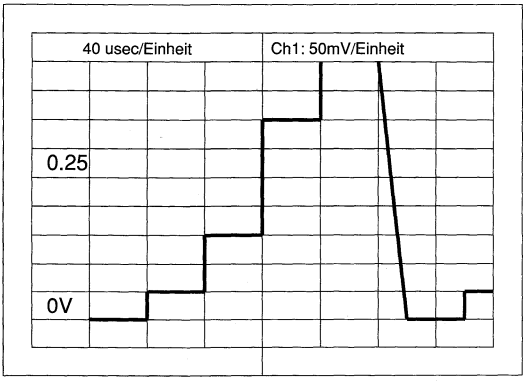


Bild 8.17: Kurvenverlauf der Ausgangsspannung des Funktionsgenerators

Programm RAMPE:

```
          ; Demonstrationsprogramm zur Ausgabe einer Rampenfunktion
          ;
          ; Deklaration der Ports
          ;
port1     equ  16385798
port2     equ  16384512
port3     equ  16415232
port4     equ  16384000
port5     equ  16385536
port6     equ  16390656
          ;
          ; Rampenprogramm
          ;
prog_io:  move.l    port1,d4
          move.l    port2,d4
          move.l    port3,d4
          move.l    port4,d4
          ;
ramp:     move.l    port5,d4
          move.l    port6,d4
          move.l    port4,d4
          move.l    port5+10,d4
          move.l    port6,d4
```

```
move.l    port4,d4
move.l    port5+20,d4
move.l    port6,d4
move.l    port4,d4
move.l    port5+50,d4
move.l    port6,d4
move.l    port4,d4
move.l    port5+120,d4
move.l    port6,d4
move.l    port4,d4
bra  ramp
```

Das Assemblerprogramm ist auf Geschwindigkeit optimiert. So verzichtet es insbesondere auf alle systemeigenen Routinen, die den zeitlichen Ablauf stark belasten. Es ist weder unterbrechbar, noch ist die Kodierung optimal in Unterprogrammen abgelegt.

Die vorangegangenen Programmbeispiele lassen sich beliebig erweitern. So bleibt es dem Anwender überlassen auch sinusförmige oder trapezförmige Ausgangssignale zu erzeugen.

8.9 Regelbare Spannungsquelle

Zu den wichtigsten Laborgeräten der Elektronik zählen Netzgeräte, die eine konstante, regelbare Ausgangsspannung erzeugen. Gute Netzgeräte geben unabhängig von der momentanen Belastung eine gleichbleibende Ausgangsspannung ab. Derartige Spannungsquellen stellen einen variablen Batterieersatz dar, wobei der Innenwiderstand nur noch wenige Milliohm beträgt. Solche konstanten Netzteile eignen sich zur Versorgung von Logikfamilien (z.B. TTL, wobei die Spannung auf exakt 5,0 V gehalten werden muß), zum Anschluß von Glühlampen (die je nach Betriebstemperatur unterschiedlich Ströme aufnehmen) oder zum Testen von Schaltungen, die trotz ihrer eigenen Stromschwankungen oder externer Spannungsschwankungen mit einer konstanten Spannung versorgt werden müssen.

Auf dem Markt sind zahlreiche Netzgeräte mit den vorher genannten Eigenschaften erhältlich. In der Regel findet man jedoch nur Laborgeräte, bei denen man die Spannung über ein Potentiometer einstellen kann. Selbst wenn die Ausgangsspannung über ein Digitalvoltmeter kontrolliert wird, fällt das Einstellen einer gewünschten Ausgangsspannung häufig recht schwer. So führen schnelle Korrekturen am Potentiometer oder das Einschalten des Geräts in vielen Fällen zu kurzzeitigen Überspannungen, die den angeschlossenen Verbraucher gefährlich werden können.

Erheblich problemloser sind dagegen programmierbare Netzgeräte, bei denen die gewünschte Ausgangsspannung digital gewählt werden kann. Eine derartige hochstabile Versorgungseinheit läßt sich mit Hilfe der Interface-Karte realisieren. Man benötigt dazu den Aufbau der

Karte als DAC nach Kapitel 8.8. Die Interface-Karte erzeugt eine programmierbare Ausgangsspannung von 0 bis 2,5 V, die an dem Interface-Stecker zwischen den Anschlußpunkten Masse und dem Analog-Ausgang abgegriffen werden kann. Bild 8.18 zeigt eine Schaltung, die an die Interface-Karte angeschlossen wird und Ausgangsspannungen von 0 bis 25,5 V erzeugt.

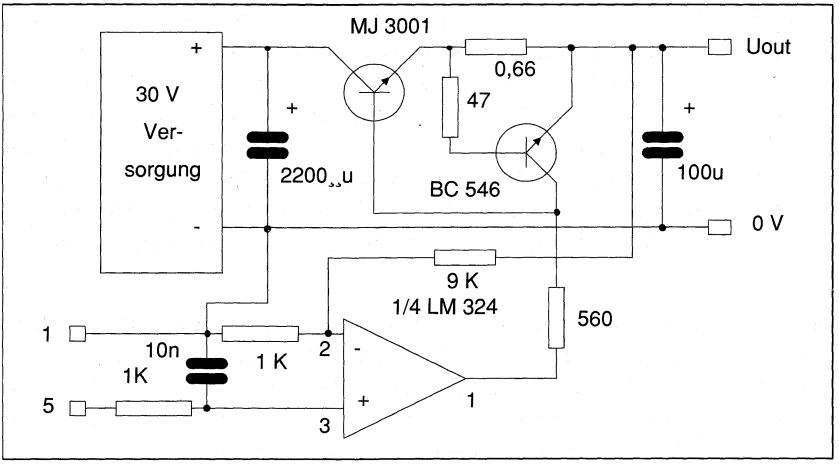


Bild 8.18: Schaltbild einer geregelten Spannungsquelle

Da das Netzteil des Atari ST keine Versorgungsspannung abgibt, die bis an diesen maximalen Pegel heranreicht, muß die externe Schaltung durch eine Versorgungsspannung von 30 V betrieben werden. Diese Versorgung baut man sich in der einfachsten Form durch einen Transformator auf, der sekundärseitig über einen Brückengleichrichter und einem Kondensatorblock eine Gleichspannung abgibt. Geeignet wäre ein Transformator mit einer Sekundärwicklung, die ca. 22 V erzeugt. Als Ladekondensator verwendet man neben dem bereits vorhandenen (2200 µF) einen Elektrolytkondensator von 10000 µF/40V. Gleichgültig welche Lösung man zur Versorgung vorsieht, es sollten auf jeden Fall einige Sicherheitsmaßnahmen beachtet werden: Die Versorgung muß von allen möglichen Erd- oder Netzpotentialen getrennt werden. Eventuelle Verkopplungen mit fremden Spannungspegeln können zur Zerstörung des Atari ST oder zu lebensgefährlichen Verletzungen von Personen führen. Man darf niemals während des Netzbetriebs Schaltungsveränderungen oder Lötungen durchführen! Die Versorgungseinheit sollte möglichst von ihrem Testaufbau separiert sein. Am besten baut man die Einheit in ein abgeschlossenes, geerdetes Gehäuse ein. Mit der Ausgangsspannung von 30 V kann man dann relativ gefahrlos experimentieren.

Zurück zur Funktionsweise der Schaltung: Am positiven Pol der Versorgungseinheit befindet sich der Darlingtontransistor MJ 3001, der über seine Basisspannung mehr oder weniger aufgesteuert werden kann. Diese Basisansteuerung bewirkt der Operationsverstärker LM

324, der die DAC-Spannung mit der Netzteil-Ausgangsspannung vergleicht. Dazu wird die Ausgangsspannung nicht direkt zugeführt, sondern nur der zehnte Teil benutzt (Spannungsteilerverhältnis: 9k zu 1k). Der Operationsverstärker verändert nur dann seine Ausgangsspannung nicht, wenn die Netzteilspannung der zehnfachen DAC-Spannung entspricht (der OPA ist dann abgeglichen). Eventuelle Eingangs- oder Lastveränderungen bewirken eine sofortige Abweichung der Vergleichspegel, und der OPA führt eine Nachregelung über den 560 Ohm-Widerstand an der Basis des MJ 3001 durch. Um eventuelle Schwingungen oder Störeinflüsse beim Leitungsanschluß am Stecker zu vermeiden, wird die DAC-Spannung nochmals vor dem OPA gesiebt (1k und 10nF). Der zweite Transistor BC 546 begrenzt den möglichen Ausgangsstrom auf ca. 1 A. Bei Strömen über 1 A fallen am Längswiderstand mehr als 0,7 V ab, die den Transistor BC 546 zum Durchschalten veranlassen. Dieses Durchschalten wirkt dem Steuerausgang des OPAs entgegen, und der MJ 3001 wird solange zugeregelt, bis der Ausgangsstrom unter den Wert von 1 A fällt. Eventuelle Überlastungen oder Kurzschlüsse können deshalb weder der Schaltung noch dem Verbraucher gefährlich werden.

Zum praktischen Aufbau der Schaltung lassen sich – je nach Verfügbarkeit – unterschiedliche Bauteiltypen einsetzen. Der Leistungs-Darlingtontransistor (ähnlich MJ 3001) sollte einen Mindeststrom vom 4 A bei einer Spannung von 60 V verkraften. Die Stromverstärkung sollte größer als 500 sein. Es eignen sich deshalb alle NPN-Typen, die für höhere Leistung bei großer Stromverstärkung ausgelegt sind. Der Transistor sollte in jedem Fall auf einen hinreichend großen Kühlkörper montiert werden. Die Gesamtverlustleistung kann bis zu 30 W betragen. Besonderer Wert ist auf die Auswahl der Spannungsteilerwiderstände (1k/9k) zu legen. Diese beiden Widerstände sollten sehr genau sein und eine geringe Temperaturdrift aufweisen. Sofern man den Wert des 9 k Widerstandes auf 9,2 k erhöht, erreicht man einen Ausgangsspannungsbereich von 0 bis 25,5 V. Die kleinste digitale Quantisierung des DACs entspricht dann genau 100 mV. In diesem Fall läßt sich jede Ausgangsspannung (in dem angegebenen Bereich) auf 100 mV genau vorprogrammieren. Andere Widerstandskombinationen erzeugen entsprechend unterschiedliche Spannungsbereiche. Bei höheren Spannungsanforderungen sollten man jedoch beachten, daß der Operationsverstärker LM 324 nur eine Maximalspannung von 32 V verträgt. Zum praktischen Aufbau reicht eine Belastbarkeit der Widerstände von 1/2 W. Der Widerstand zur Stromüberwachung (0,66 Ohm) sollte mindestens 1 W vertragen.

Zur Programmierung der Spannungsquelle reicht ein kurzes Basic-Programm. Es ist ähnlich aufgebaut wie das Programm im Kapitel 8.8.

8.10 Regelbare Stromquelle für geerdete Verbraucher

Der Einsatz von Stromquellen ist weit weniger verbreitet als der von Spannungsquellen. Dennoch gibt es vielfältige Anwendungsbereiche, bei denen man nicht an einer konstanten Spannung, sondern an einem konstanten Strom interessiert ist. Zu den wichtigsten Einsatzbereichen von Konstantstromquellen gehören die Messung von Widerständen und das Erzeugen von linearen Rampen an Kondensatoren. Bei der Widerstandsmessung wird der zu prüfende Verbraucher mit einem konstanten Strom versorgt. Nach dem Ohmschen Gesetz ist der Spannungsabfall proportional zum Widerstandswert. Man erhält so eine lineare Widerstands/Spannungs-Skala. Je nach Wahl des Konstantstroms lassen sich besonders kleine Widerstandswerte sehr genau bestimmen. Zum Erzeugen einer linearen Spannungsrampe beschickt man einen entladenen Kondensator mit einem Konstantstrom. Da die Spannungsänderung am Kondensator proportional zu seiner Ladungsänderung ist, entsteht am Kondensator eine linear ansteigende Spannung, die nur vom Kondensatorwert und vom Versorgungsstrom abhängt.

Neben diesen beiden wichtigen Anwendungsbereichen gibt es zahlreiche Sensoren für Druck, Temperatur usw., die einer konstanten Stromversorgung bedürfen.

Die Schaltung nach Bild 8.19 ist in der Versorgungsumgebung ähnlich aufgebaut wie die im vorigen Kapitel beschriebene Spannungsquelle.

Alle notwendigen Details, die Aufbauhinweise und die Vorsichtsmaßnahmen gelten auch hier in gleicher Form. Der Operationsverstärker wird jedoch – im Gegensatz zur Konstantspannungsquelle – nicht zum Vergleich der geregelten Ausgangsspannung benutzt, sondern dient zum Konstanthalten eines Steuerstroms. Hierzu vergleicht der OPA LM 324 die vom DAC kommende Steuerspannung mit dem Spannungsabfall an dem 100-Ohm-Widerstand R2. Der Transistor T1 wird solange im Steuerstrom verändert, bis der Emitterstrom einen Spannungsabfall an R2 erzeugt, der dem Eingangswert entspricht. Die Eingangsspannung U1 ist damit identisch zum Spannungsabfall U2. Der Emitterstrom des Transistors kann somit von 0 bis 25 mA eingestellt werden. Da sich dieser Emitterstrom aus dem Kollektor- und dem Basisstrom des Transistors zusammensetzt, ist der Kollektorstrom I3 nur dann dem Emitterstrom I2 gleich, wenn der Transistor eine hohe Stromverstärkung aufweist. Je niedriger die Stromverstärkung des verwendeten Transistors ist, desto mehr geht der Basisstrom als Fehler in die Funktion dieser Stromquelle ein.

Die Schaltung des OPAs mit dem Steuertransistor T1 stellt bereits eine brauchbare Stromquelle dar, wenn die Polarität des Stromflusses keine Rolle spielt. Für einfache Sensoren erfüllt diese Version bereits ihren Zweck. In vielen Fällen benötigt man jedoch eine Stromquelle, die einen geerdeten Verbraucher versorgen kann. Die Stromflußrichtung muß für diese Anwendungen umgedreht werden. Zur Stromumkehr kann man einen sogenannten Stromspiegel benutzen, der in der Schaltung durch zwei Transistoren realisiert wird. So

erzeugt der Kollektorstrom von T1 am Widerstand R3 einen Spannungsabfall, der genau dem von U2 (also auch U1) entspricht. Der Transistor T2 ist als Diode geschaltet und dient lediglich als potentialverschiebendes Element der Basis-Emitter-Spannung. Da sich die beiden Transistoren T2 und T3 absolut identisch verhalten, steuert der Transistor T2 den Transistor T3 so weit an, bis der Spannungsabfall U4 dem Wert von U3 entspricht. Damit wird ein Stromfluß erreicht (I5), der genau dem Vorgabestrom I2 gleich ist. Da sich jedoch die Stromflußrichtung geändert hat, lassen sich auch geerdete Verbraucher anschließen.

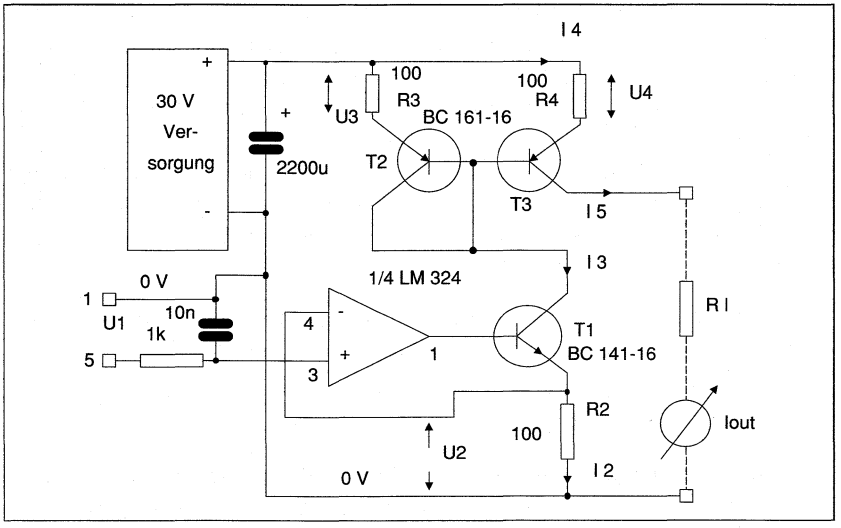


Bild 8.19: Schaltbild einer geregelten Stromquelle für geerdete Verbraucher

Die Schaltung erzeugt bei allen möglichen Belastungen einen hochkonstanten Strom. Man kann sie vom Kurzschluß bis zu einer Ausgangsspannung von ca. 25 V betreiben. Die Widerstandswerte der Widerstände (3x100 Ohm) müssen sehr genau sein und dürfen sich mit der Temperatur kaum verändern. Die Transistoren müssen eine hohe Stromverstärkung aufweisen und über gleichartige Eigenschaften verfügen (insbesondere T2 und T3). Die Versorgung des OPAs wurde im Schaltbild weggelassen – sie erfolgt über die Anschlüsse 11 und 4 zur 30-V-Versorgung. Je nach Anwendungsbereich läßt sich der mögliche Stromwert über die Widerstände variieren. Bei hohen Strömen müssen entsprechende Leistungstransistoren, bei extrem kleinen Feldeffekttransistoren eingesetzt werden.

Das benötigte Programm zur Ansteuerung ist nahezu identisch zum dem für die Spannungsquelle.

8.11 Interface für analoge Eingabe

Für die analoge Ausgabe wurde nur ein Teil des LM 324 (nämlich nur zwei der vier Operationsverstärker) benutzt. Ein weiterer OPA diente bereits zur Konversion analoger Eingabegrößen. Dazu werden verschiedene Wandelungsverfahren verwendet, die bei den ADC-Typen bereits üblich sind. Der dritte Operationsverstärker vergleicht dabei die analoge Eingangsspannung mit dem analogen Ausgabewert. Er ist selbst als Komparator mit einer Hysterese von 1/1000 (1K/1M) beschaltet. Sein Ausgang zeigt dem Baustein 8255 (am Eingang C0) an, ob der momentane analoge Ausgabewert oder die Eingangsspannung größer ist.

Die im Kapitel 6 beschriebenen Konversionsverfahren lassen sich auch ohne die Integration eines ADCs realisieren. Zu ihnen gehören:

- ADC nach dem Rampenverfahren
- ADC nach der Methode der sukzessiven Approximation

Die beiden folgenden Kapitel beschäftigen sich mit diesen ADC-Typen und stellen die notwendige Software vor.

8.12 Interface für ein ADC nach dem Rampenverfahren

Beim Rampenverfahren erzeugt der DAC-Teil des ADCs eine linear ansteigende Rampe, die schrittweise Spannungen von 0 bis 2,5 Volt mit der zu messenden Eingangsspannung vergleicht. Ein Komparator zeigt an, ob die Rampenspannung oder die Eingangsspannung größer ist und teilt der Logikeinheit das Ergebnis mit. Die beiden folgenden Aufbauten lassen sich mit einem Potentiometer von 1 kOhm testen, wenn die Endpunkte zwischen den Anschlüssen 7 (Referenzspannung) und 1 (Masse) liegen und der Mittelschleifer mit dem Anschluß 3 (ADC-Eingang) verbunden ist. Das folgende Basic-Programm programmiert das ADC nach dem Rampenverfahren und mißt in dieser Form die Höhe der Eingangsspannung:

Programm R_ADC

```
REM ***********************************************
REM *        ADC nach dem Rampenverfahren         *
REM ***********************************************
REM
GOSUB prog_io
REPEAT
  a=0
  REPEAT
    GOSUB out_a
```

```
    a=a+1
    GOSUB read_c
  UNTIL ODD(c) OR a=256
  PRINT " Digitalwert: ",a-1
UNTIL INKEY$=CHR$(&H1B)
END
REM
PROCEDURE prog_io
  x=PEEK(16385798)
  x=PEEK(16384512)
  x=PEEK(16415232)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE out_a
  x=PEEK(16385536+2*a)
  x=PEEK(16390656)
  x=PEEK(16384000)
RETURN
REM PROCEDURE out_c
  x=PEEK(16385536+2*c)
  x=PEEK(16407040)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE read_c
  c=DPEEK(16404992)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE print_out
  GOSUB read_c
  IF c=65523
    y=1
  ENDIF
  IF c=65535
    y=2
  ENDIF
  GOSUB out_a
  c=127
```

```
  GOSUB out_c
  c=255
  GOSUB out_c
RETURN
```

Das Programm beginnt – nach erfolgter Programmierung des 8255 – mit dem Aufbau der Rampe am DAC. Der Vergleichswert am Komparator wird am Eingang des 8255 abgefragt. Je nach Vergleichswert erfolgt danach ein erneutes Erhöhen der Rampe oder die Ausgabe der Ausgangsspannung. Die auf dem Bildschirm ausgegebenen Werte befinden sich im Zahlenbereich von 0 bis 255 und entsprechen den Spannungswerten von 0 bis 2,5 Volt.

Eine leichte Änderung des vorher aufgeführten Basic-Programms bewirkt eine direkte Spannungsanzeige auf dem Bildschirm:

```
REM ***********************************************
REM *        ADC nach dem Rampenverfahren         *
REM ***********************************************
REM
GOSUB prog_io
REPEAT
  a=0
  REPEAT
    GOSUB out_a
    a=a+1
    GOSUB read_c
  UNTIL ODD(c) OR a=256
  PRINT " Digitalwert: ",(a-1)*0.00982
UNTIL INKEY$=CHR$(&H1B)
END
REM
PROCEDURE prog_io
  x=PEEK(16385798)
  x=PEEK(16384512)
  x=PEEK(16415232)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE out_a
  x=PEEK(16385536+2*a)
  x=PEEK(16390656)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE out_c
```

```
  x=PEEK(16385536+2*c)
  x=PEEK(16407040)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE read_c
  c=DPEEK(16404992)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE print_out
  GOSUB read_c
  IF c=65523
    y=1
  ENDIF
  IF c=65535
    y=2
  ENDIF
  GOSUB out_a
  c=127
  GOSUB out_c
  c=255
  GOSUB out_c
RETURN
```

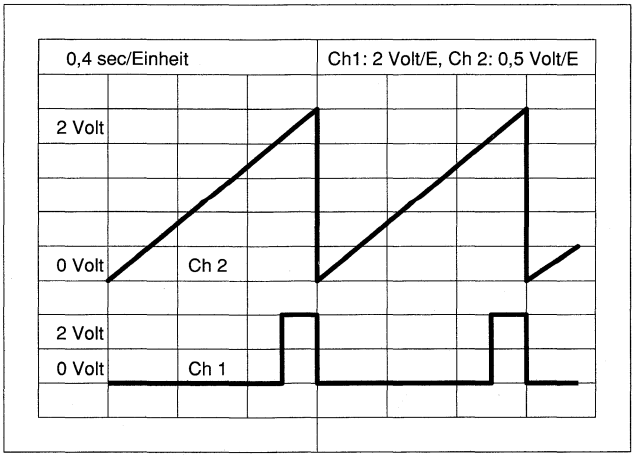


Bild 8.20: Zeitlicher Verlauf der Spannung beim Rampen-ADC

Bild 8.20 zeigt den zeitlichen Ablauf bei dem Rampen-ADC in der Darstellung nach einem Oszilloskopen-Bild. Dabei wurde im Kanal 2 die Rampenspannung und im Kanal 1 die Komparatorschwelle für eine Spannung von 2 Volt dargestellt.

8.13 Interface nach der Methode der sukzessiven Approximation

Ersetzt man den inkrementellen Modus im Programm zur Ansteuerung des 8255 durch ein Wichtungsverfahren, so erhält man eine Konversion nach der Methode der sukzessiven Approximation. Wie im Kapitel 6 beschrieben, wird bei diesem Verfahren die Eingangsspannung durch geschicktes Anlegen der Referenzgröße erraten. Die Programmierung entspricht dieser Vorgehensweise bei der fortlaufenden Halbierung der Referenzspannung mit Addition oder Subtraktion. Daraus erhält man das folgende Basic-Programm:

Programm S_ADC

```
REM ***********************************************
REM *    ADC nach der Methode der sukz. Appr.     *
REM ***********************************************
REM
GOSUB prog_io
REPEAT
  r=128
  z=8
  a=0
  REPEAT
    a=a+r
    GOSUB out_a
    GOSUB read_c
    IF ODD(c)
      a=a-r
    ENDIF
    z=z-1
    r=r/2
  UNTIL z=0
  PRINT " Digitalwert: ",a
UNTIL INKEY$=CHR$(&H1B)
END
REM
PROCEDURE prog_io
  x=PEEK(16385798)
```

```
  x=PEEK(16384512)
  x=PEEK(16415232)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE out_a
  x=PEEK(16385536+2*a)
  x=PEEK(16390656)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE out_c
  x=PEEK(16385536+2*c)
  x=PEEK(16407040)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE read_c
  c=DPEEK(16404992)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE print_out
  GOSUB read_c
  IF c=65523
    y=1
  ENDIF
  IF c=65535
    y=2
  ENDIF
  GOSUB out_a
  c=127
  GOSUB out_c
  c=255
  GOSUB out_c
RETURN
```

Für alle Werte wird immer dieselbe Konversionszeit benötigt. Für Spannungswerte unter dem Digitalwert 10 hat das Rampenverfahren noch einen Zeitvorteil, bei allen Werten ab 11 (bis 255) ist das ADC nach der Methode der sukzessiven Approximation schneller. Es empfiehlt sich deshalb, für alle weiteren Beispiele von der Methode der sukzessiven Approximation auszugehen und das letzte Basic-Programm zu verwenden. Bild 8.21 zeigt einen Spannungsverlauf am DAC bei der Ermittlung einer Meßspannung.

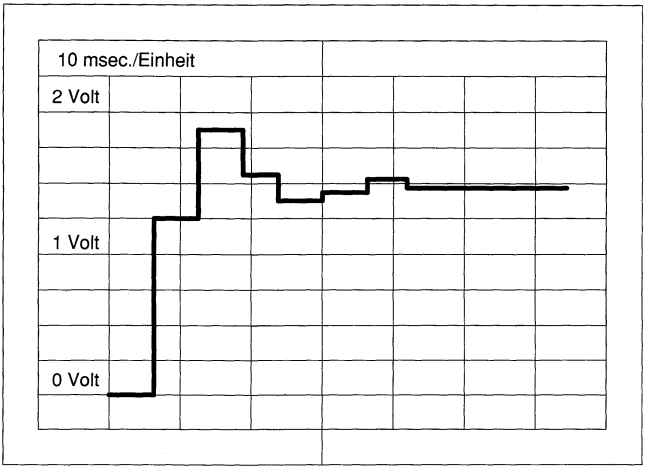


Bild 8.21: Zeitlicher Verlauf der Spannung beim ADC nach sukzessiver Approximation

8.14 Anschluß von Sensoren am Beispiel der Temperaturmessung

Anhand einer Temperaturmessung soll im folgenden der Anschluß von Sensoren an das ADC gezeigt werden. Ein Thermofühler, der seinen Widerstandswert mit der Temperatur verändert, ist ein besonders einfacher und preiswerter Sensor dafür, übliche Umgebungstemperaturen zu messen. Ein derartiger Thermofühler besteht aus einem NTC-Widerstand (Negative-Temperature-Coefficient) im Widerstandsbereich um 10 kOhm. Die nachfolgend aufgeführten Programme beziehen sich auf einen NTC mit 6,8 kOhm (z.B. Siemens K 164/6K8). Man schließt den Widerstand zwischen der Referenzspannung (Flachsteckerpol 7) und dem ADC-Eingang (Flachsteckerpol 3) an. Zwischen dem ADC-Eingang und Masse (Flachsteckerpol 1) sorgt ein Ableitungswiderstand von 3,9 kOhm zur richtigen Spannungsanpassung (Bild 8.22).

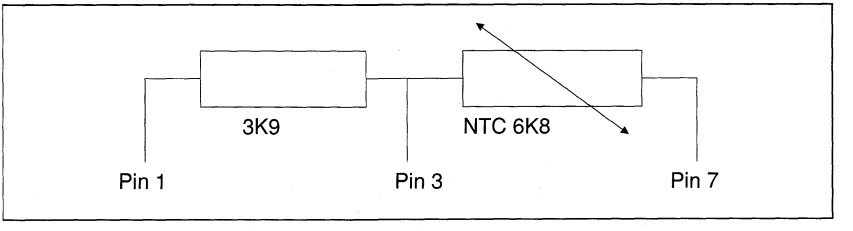


Bild 8.22: Beschaltung des AD-Wandlers

NTC-Widerstände verändern ihren Widerstandswert nach der Formel:

$$R(T) = R(n) * e^{B\left(\frac{1}{T} - \frac{1}{Tn}\right)}$$

Dabei sind die Größen:

R(T): Widerstand bei der Temperatur T
R(n): Widerstand bei der Temperatur von 293,15° K (20° C)
B: Materialkonstante des NTCs
T: Temperatur in °K
Tn: 293,15° K

Bei allen weiteren Berechnungen wird von der Kelvin-Skala (K) ausgegangen. Bei dieser Skala wählt man den absoluten Nullpunkt bei –273,15° Celsius. Man erhält die Celsius-Skala, indem man von der Kelvin-Skala 273,15 abzieht.

Zum Eichen des NTC-Thermometers wird das ADC-Programm nach der sukzessiven Approximation benutzt und die jeweiligen Werte für 0° C und für 100° C bestimmt. 0° C läßt sich mit Eiswasser erzeugen. Im Gleichgewichtszustand zwischen Wasser und Eis liegen genau 0° C vor. 100° C erhält man in kochendem Wasser. Das folgende Basic-Programm zeigt eine Anpassung an einen NTC mit dem Wert von 6,8 kOhm (nur noch teilweise dargestellt).

```
.     z=z-1
    r=r/2
  UNTIL z=0
  PRINT " Digitalwert: ",(a-30)/1.65
 UNTIL INKEY$=CHR$(&H1B)
 END
```

Dieses Programm gibt die Temperaturwerte im Bereich von –10° bis 120° C mit einer Genauigkeit von ca. 5° C wieder. Dieses relativ gute lineare Verhalten läßt sich folgendermaßen erklären: Die Meßspannung Us ergibt sich aus dem Spannungsteilerverhältnis zwischen dem Widerstandswert des NTCs R(T) und dem Festwiderstand R von 3,9 KOhm, gemessen an der Referenzspannung.

$$Us = \frac{U\ ref}{1 + \frac{R(T)}{R}}$$

Der NTC-Widerstand verändert seinen Wert mit der Exponential-Funktion. Diese Exponential-Funktion läßt sich jedoch bei kleinen Veränderungen entwickeln (nach Taylor). Bei dieser Entwicklung ersetzt man den wahren Wert der Funktion durch eine Summe. Die

Exponential-Funktion mit dem Exponenten x kann in einfachster Näherung auch als 1 + x geschrieben werden. Daraus ergibt sich aus der Formel für den NTC-Widerstand die einfache Näherung:

$$R(T) = Rn * \left(1 + B\left(\frac{1}{T} - \frac{1}{Tn}\right)\right)$$

Setzt man diese Näherung in die obere Formel für die Meßspannung ein, erhält man eine ungefähr lineare Beziehung zwischen dem Meßwert und der Temperatur. Es bleibt dem Geschick des Experimentators überlassen, eventuelle Verbesserungen an den Parametern des Programms vorzunehmen, um eine bessere Linearität zu erhalten. In kleinen Bereichen (z.B. von 15° C bis 30° C) vermag das Basic-Programm auch Temperaturen mit einem Fehler von weniger als einem Grad genau anzugeben.

Möchte man diese Linearität im gesamten Temperaturbereich erhalten, muß man in dem Basic-Programm eine genaue Berechnung des Temperaturwerts durchführen. Man setzt hierzu die Formel für den NTC-Widerstandswert in die Meßspannungsformel ein und logarithmiert diese. Mit der Materialkonstante B von ca. 4200 ergibt sich nach einiger Umrechnung folgende Funktion für die Temperatur (in Kelvin):

$$T = \frac{-4200}{\ln Us - \ln(2{,}5 - Us) - 13{,}54}$$

Diese Formel kann man direkt in das Basic-Programm einsetzen und erhält damit eine optimale Anpassung an das Verhalten des NTC-Widerstands (nur noch der entsprechende Teil dargestellt).

```
.     ENDIF
    z=z-1.
    r=r/2
  UNTIL z=0
  u=a/102.4
  PRINT " Digitalwert:",(-4200/(log(u)-log(2.5-u)-13.54))-273.15
UNTIL INKEY$=CHR$(&H1B)
END
```

Dieses Basic-Programm unterscheidet sich von dem vorhergehenden lediglich durch zwei Zeilen. So wurde in diesen Zeilen eine Umrechnung zwischen dem Digitalwert und der Meßspannung nach der NTC-Formel eingeführt.

Die Anpassung des ADCs mit dem entsprechenden Programm für einen NTC-Widerstand zur Temperaturmessung ist nur eines von vielen Beispielen, in welcher Form sich analoge Größen im Rechner verarbeiten lassen.

8.15 Speicheroszilloskop für analoge Signale

Die nächsten vier Kapitel beschäftigten sich mit dem Erfassen von analogen Meßwerten. Die Beispiele waren dabei noch recht einfach und die kurzen Programme dienten der Meßwertaufnahme und der alphanumerischen Ausgabe von Spannungs- oder Temperaturwerten. Mit der Interfacekarte lassen sich aber auch wesentlich komplexere Aufgaben lösen, zum Beispiel der Aufbau eines Speicheroszilloskops, mit dessen Hilfe die Darstellung langsam veränderlicher Größen möglich ist. Bei diesem Beispiel werden die Meßwerte grafisch dargestellt. Das dafür benötigte Basic-Programm ist etwas umfangreicher. Zusammen mit der Interfacekarte (und dem Aufbau als ADC) können mit dem Programm analoge Spannungswerte im Bereich von 0 bis 2,5 V gemessen werden. Dabei entnimmt das ADC in festgelegten Zeitabschnitten die Meßwerte und stellt sie grafisch als Funktion der Zeit dar. Ein Meßzyklus umfaßt 550 Werte.

Das Programm erlaubt die Wahl von beliebigen Zeiteinheiten von 20 bis 999 ms pro Speicherung. Von der Bedienfunktion ist es ähnlich aufgebaut wie das Programm zum Logikanalysator. Unterschiedlich ist jedoch die multifunktionale Darstellung, bei der mehrere Kurvenformen übereinander darstellbar sind.

Das Programm hat folgende Form:

Programm S_SCOPE

```
REM ***********************************************
REM *  Demonstrationsprogramm eines Speicherskops *
REM ***********************************************
REM
REM ***********************************************
REM *                Hauptprogramm                *
REM ***********************************************
REM
t=20
REPEAT
  GOSUB prog_io
  GOSUB screen
  REPEAT
    x$=INKEY$
    IF x$="h"
      HARDCOPY
    ENDIF
    IF x$="s"
      GOSUB graph
    ENDIF     IF x$="t"
```

```
      GOSUB time
    ENDIF
  UNTIL x$="q" OR x$=CHR$(&H1B)
  IF x$="q"
    END
  ENDIF
UNTIL x$="q"
END
REM ***********************************************
REM *                Unterprogramme               *
REM ***********************************************
REM
PROCEDURE screen
  CLS
  DEFLINE 1,3,0,0
  BOX 5,5,635,395
  PRINT AT(30,19);" Analoges Speicheroszilloskop"
  DEFLINE 1,1,0,0
  BOX 20,280,500,390
  PRINT AT(4,19);" Bedienfunktionen: "
  PRINT AT(6,20);" Löschen:    ESC "
  PRINT AT(6,21);" Starten:    S "
  PRINT AT(6,22);" Zeitbasis:  T "
  PRINT AT(6,23);" Hardcopy:   H "
  PRINT AT(6,24);" Pr.-Ende:   Q "
  PRINT AT(30,23);" Zeit (msec):    "
  DEFLINE 2,1,0,0
  LINE 50,270,600,270
  LINE 50,168,600,168
  LINE 50,65,600,65
  LINE 50,14,600,14
  DEFLINE 3,1,0,0
  y=50
  REPEAT
    LINE y,14,y,270
    y=y+50
  UNTIL y>660
  PRINT AT(3,2);"2.5"
  PRINT AT(3,7);"Volt"
  PRINT AT(3,5);"2.0"
  PRINT AT(3,11);"1.0"
```

```
  PRINT AT(3,17);"0.0"
  PRINT AT(67,19);USING "###.##",t/2
  PRINT AT(73,19);" sec"
RETURN
REM
PROCEDURE time
  REPEAT
    PRINT AT(44,23);"        "
    LOCATE 23,44
    INPUT t
    IF t<20  OR t>999
      GOSUB error
    ENDIF
  UNTIL t>=20 OR t<=999
  PRINT AT(30,24);"                        "
  t=t/20
  t=INT(t)*20
  PRINT AT(67,19);USING "###.##",t/2
  PRINT  AT(73,19);" sec" RETURN
REM
PROCEDURE error
  PRINT AT(30,24);" Bereich:20-999!"
  PAUSE 100
RETURN
REM
PROCEDURE graph
  DEFLINE 1,1,0,0
  y=50
  REPEAT
    GOSUB suk_adc
    PLOT y,270-a
    y=y+1
    PAUSE (t/20-1)
  UNTIL y=600
RETURN
REM
PROCEDURE suk_adc
  r=128
  z=8
  a=0
```

```
  REPEAT
    a=a+r
    GOSUB out_a
    GOSUB read_c
    IF ODD(c)
      a=a-r
    ENDIF
    z=z-1
    r=r/2
  UNTIL z=0
RETURN
REM
PROCEDURE prog_io
  x=PEEK(16385798)
  x=PEEK(16384512)
  x=PEEK(16415232)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE out_a
  x=PEEK(16385536+2*a)
  x=PEEK(16390656)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE out_c
  x=PEEK(16385536+2*c)
  x=PEEK(16407040)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE read_c
  c=DPEEK(16404992)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE print_out
  GOSUB read_c
  IF c=65523
     y=1
  ENDIF
```

```
  IF c=65535
    y=2
  ENDIF
  GOSUB out_a
  c=127
  GOSUB out_c
  c=255
  GOSUB out_c
RETURN
```

Mit dem Speicheroszilloskop lassen sich veränderliche Vorgänge recht gut darstellen, so daß es für langsame Vorgänge gegenüber einem Oszillographen erhebliche Vorteile aufweist. Bild 8.23 zeigt die Aufnahme zweier Sinus-Signale, wie sie von einem Funktionsgenerator erzeugt werden. Beide Kurvenformen wurden nacheinander aufgenommen und grafisch dargestellt.

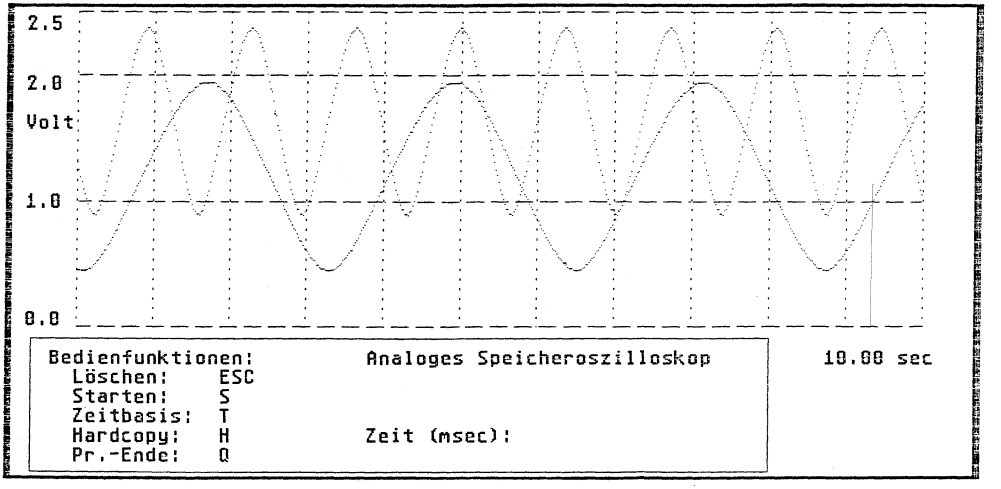


Bild 8.23: Hardcopy einer Meßwerterfassung beim analogen Speicheroszilloskop

8.16 Zweipunktregler für Temperaturstabilisierung

Die Bewegungsvorgänge unserer Umwelt laufen nahezu ausnahmslos in geregelten Prozessen ab. Sowohl einfache motorische Vorgänge bei Lebewesen wie Bewegen der Arme und Beine, als auch komplexer Bewegungsabläufe wie Gehen oder Greifen, erfolgen nach festgelegten Regelmechanismen. Seit mechanische Geräte oder elektronische Einrichtungen die Arbeit des Menschen unterstützen, müssen auch im Maschinenablauf Regelvorgänge integriert werden. Diese Regelvorgänge erstrecken sich von einfachsten Anwendungen bei Heizungssteuerungen oder Wasserstandsreglern bis zu komplexen Aufgaben, wie die Positionierung von Robotern. Um in die Problematik von Regelaufgaben einzusteigen, werden die Regelgrößen und die Wirkungsweise von Regelvorgängen anhand eines einfachen Beispiels erklärt: Die Aufgabe besteht darin, den Wasserstand in einem Becken stets auf konstanter Höhe zu halten (Bild 8.24).

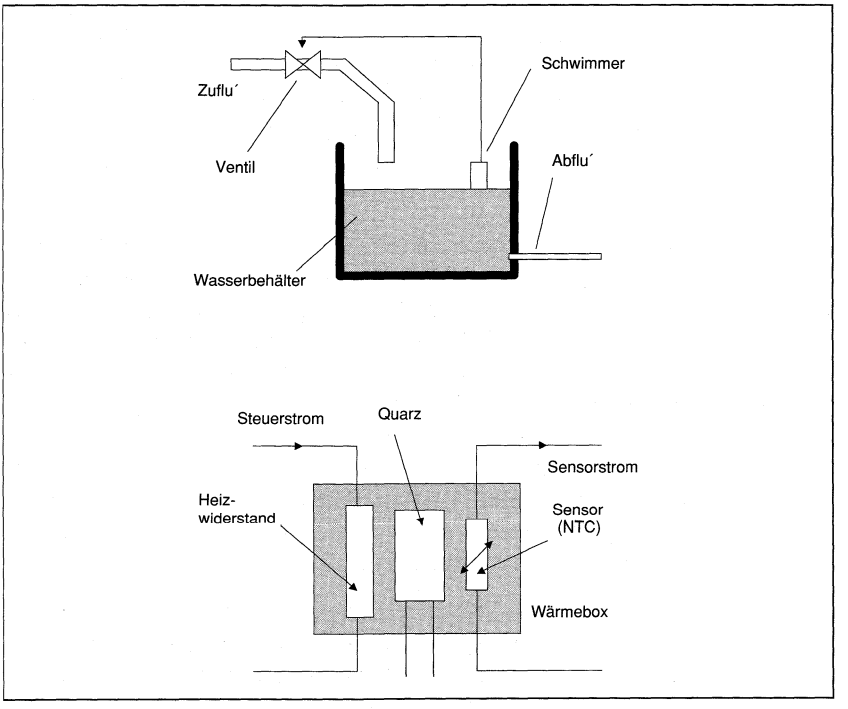


Bild 8.24: Prinzipieller Aufbau eines geregelten Wasserbehälters

Die Aufgabe des Regelsystems besteht darin, immer genau soviel Wasser nachzuliefern, wie zum Aufrechterhalten des Wasserstands notwendig ist. Beim Regler wird der gewünschte

Wasserstand als Sollwert und der tatsächlich vorhandene als Istwert bezeichnet. Das Regelsystem hat die Aufgabe, Soll- und Istwert einander anzugleichen. Der Abfluß (oder ein eventuelles Verdampfen von Wasser) wirken als Störgrößen des Systems. Der Regler wirkt diesen Störgrößen entgegen, indem er aus der Differenz des Soll- und Istwerts eine Stellgröße ableitet, die den Zufluß regelt.

Auf den ersten Blick scheint diese Regelaufgabe leicht zu lösen zu sein. So könnte man einen Schwimmer an der Wasseroberfläche anbringen und beim Unterschreiten des gewünschten Wasserstands das Zuflußventil öffnen. Sobald der Wasserstand wieder stimmt, schließt man das Ventil. Dabei wird aber nur eine ungefähre Angleichung durchgeführt, da mit diesem Verfahren die Wasserhöhe stets deutlich überschritten wird. Sobald sich nämlich ein Gleichstand zwischen Soll- und Istwert eingestellt hat, wird der Zufluß gesperrt. Das Zulaufrohr ist jedoch in diesem Moment noch mit Wasser gefüllt, das sich im Becken sammelt. Der Istwert wird also laufend zwischen dem Sollwert und einem höheren Wert (der von dem Rohrvolumen abhängt) hin- und herpendeln. Gleiches Verhalten stellt sich beim Regler auch ein, wenn die Reaktion auf die Stellgröße nicht unendlich schnell erfolgt, sondern mit einem Zeitverzug einsetzt. In diesem Fall machen sich sogenannte Totzeiten beim Regelverhalten bemerkbar. Die Zeitdifferenz, die zwischen dem Erkennen der Regelabweichung und der Reaktion (Öffnen des Ventils) verstreicht, wird als Einschalttotzeit bezeichnet. Manchmal haben Regler auch eine Ausschalttotzeit, die zwischen dem Erkennen des Gleichstands und dem Schließen des Ventils verstreicht. Das lange Rohr beim Zufluß macht sich ebenfalls als Ausschalttotzeit bemerkbar.

Das Regelverhalten, bei der die Stellgröße nur zwei Positionen einnehmen kann (Ventil auf oder zu) wird auch bei noch so komplexer Berechnung aller Parameter immer eine leichte Schwankung des Wasserstands mit sich bringen. Derartige Zweipunktregler werden deshalb nur dort eingesetzt, wo man an einem im Mittelwert richtigen Istwert interessiert ist. Soll zu jeder Zeit ein Gleichstand zwischen Soll- und Istwert erzeugt werden, muß der Regler mit einer proportionalen Stellgröße arbeiten, die das Zulaufventil in Abhängigkeit von der Differenz zwischen Soll- und Istwert mehr oder weniger öffnet. Dieses Regelverhalten wird deshalb auch Proportionalregler (P-Regler) genannt. Die bereits vorgestellten Beispiele der geregelten Spannungs- bzw. Stromquelle arbeiten nach diesem Proportionalverfahren. Der verwendete Operationsverstärker regelt in beiden Fällen einen Transistor so weit, bis die gewünschte Differenz zwischen Soll- und Istwert genau 0 V beträgt.

Neben dem P-Regler ist auch noch der D-Regler (Differential-Regler) bekannt, der seine Regelgröße aus der zeitlichen Veränderung der Regelabweichung erzeugt. Manchmal ist es auch sinnvoll, die Stellgröße bei gleichbleibender Abweichung zeitlich immer weiter zu verändern. Ein derartiger Regler, der seinen Proportionalanteil zeitlich verändert, heißt I-Regler (Integral-Regler). Komplexe Regelvorgänge lassen sich häufig nur dann realisieren, wenn alle drei Regeltypen gleichzeitig kombiniert werden. Man erhält so einen PID-Regler, dessen Einzelanteile aus der Aufgabenstellung des Problems berechnet werden müssen.

Wie bereits vorher erwähnt, lassen sich einfache Regelaufgaben auch mit einem Zweipunktregler gut realisieren. Zu den typischen Aufgaben des Zweipunktreglers gehört die Temperaturregelung, wobei der Istwert mit einer kleinen Abweichung stets um den Sollwert schwanken darf. Das Beispiel des Wasserbehälters aus Bild 8.24 läßt sich sehr leicht auf einen Temperaturregler übertragen. Als Sollwert ergibt sich dann die gewünschte Temperatur. Die tatsächliche Temperatur wird durch den Istwert charakterisiert. Als Zufluß dient ein Schaltelement, das die Heizung einschaltet, und als Störgrößen sind die Konvektion und die Abstrahlung des Heizkörpers zu berücksichtigen. Für den Laboraufbau könnte eine Temperaturregelung aus einem Widerstand bestehen, der ein Fremdelement aufheizt, bis ein Temperatursensor den Sollwert bestätigt und die Abschaltung der Widerstandsversorgung vorgibt. Derartige einfache Widerstandsheizungen sind bei elektronischen Geräten weit verbreitet. Sie dienen der Temperaturkonstanthaltung von Bauelementen, die zu thermischer Drift neigen (Quarze, Transistoren oder Operationsverstärker). Bild 8.25 zeigt eine typische Wärmebox, die durch einen Zweipunktregler angesteuert werden kann.

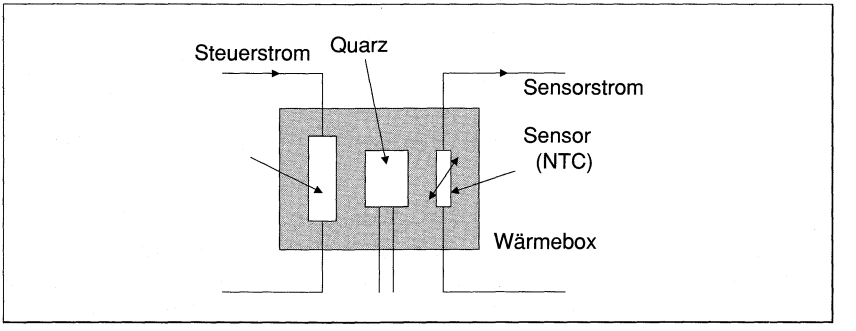


Bild 8.25: Prinzipieller Aufbau einer Wärmebox

Der Sollwert kann vom Istwert nur dann erreicht werden, wenn sich die gewünschte Temperatur (Sollwert) zwischen der Umgebungstemperatur und der höchsten Grenztemperatur befindet. Je weiter der Sollwert von den Grenzpunkten entfernt liegt, desto günstiger ist auch das Regelverhalten. In der Nähe der Grenztemperatur braucht der Regler sehr lange, bis er die Abweichung ausgleichen kann. Genauso erfolgt bei Temperaturbereichen nahe der Umgebungstemperatur kaum noch eine hinreichende Abkühlung. Bild 8.26 zeigt die möglichen Regelverhältnisse in einem Temperatur/Zeit-Diagramm.

Sobald man den Heizwiderstand mit Strom beschickt, beginnt seine Erwärmung. Die Temperaturkurve (im Bild Kurve A) verläuft nach einer Exponential-Funktion, die sich nach langer Zeit dem Maximalwert nähert. Unterhalb des Maximalwerts läßt sich die Funktion durch eine Gerade approximieren, die einen konstanten Temperaturanstieg pro Zeiteinheit aufweist. Nach dem Abschalten der Widerstandsversorgung fällt die Temperatur ebenfalls längs einer Geraden ab (im Bild Kurve A). Lediglich in der Nähe der Umgebungstemperatur

erfolgt die Angleichung an eine Exponential-Funktion. Die Anstiegs- und Abfallgeschwindigkeit hängt von der Wahl der Bauteile (Größe, Belastung usw.) und vom Aufbau des Systems ab (Fläche, Konvektion, Abstrahlung usw.).

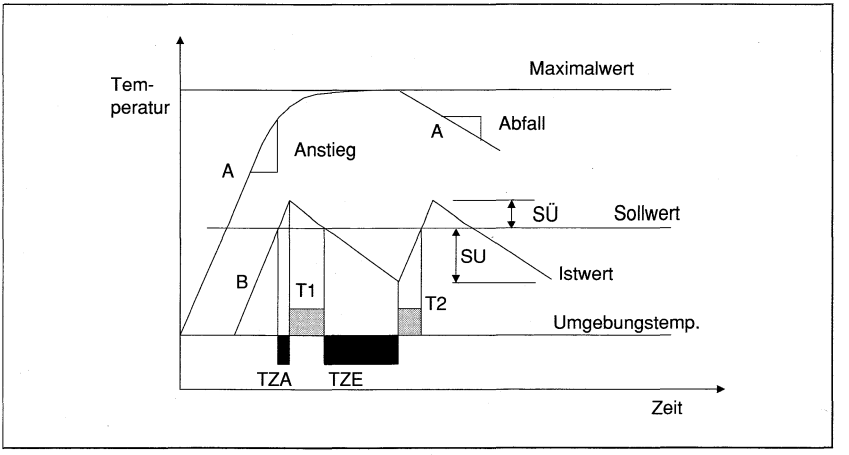


Bild 8.26: Beispiel eines Zeitdiagramms für verschiedene Regelgrößen

Im geregelten Fall (im Bild Kurve B) wird der Widerstand nur solange mit Strom versorgt, bis der Sollwert erreicht wird. Eine Sollwertüberschreitung findet genau dann statt, wenn der Heizeffekt des Widerstands noch länger anhält. Je größer die Ausschalttotzeit ist, desto mehr wird der Sollwert übertroffen. In Bild 8.26 ist eine mögliche Ausschalttotzeit (TZA) durch ein schwarzes Feld angedeutet. Nach Ablauf dieser Totzeit hat der Heizwiderstand keine Wirkung mehr und die Temperatur nimmt stetig ab (T1). Sobald der Sollwert unterschritten wird, kann eine Einschalttotzeit zur Unterschreitung der gewünschten Temperatur führen (zweites schwarzes Feld). Erst nach dem Ablauf dieser Verzögerung bewirkt der Widerstand wieder eine Erwärmung. Der Sollwert wird dann nach der Zeit (T2) erneut überschritten. Von dieser Stelle ab beginnt der Vorgang erneut. Der tatsächliche Istwert schwankt je nach Totzeit beim Ein- und Ausschalten um den Sollwert. Es wird eine laufende Sollwertüberschreitung SÜ und eine Sollwertunterschreitung SU in Kauf genommen.

Die Schwankungswerte lassen sich aus den Vorgaben leicht berechnen und ermöglichen dadurch eine optimale Einstellung der Regelparameter und der Totzeiten: Die Überschreitung des Sollwerts ergibt sich dabei aus dem Sollwerts S, aus dem Maximalwerts M, aus der Anstiegszeit bei angenommenen linearem Verlauf Ts und aus der Ausschalttotzeit Ta. Das Anstiegsdreieck ergibt eine Temperaturanstiegsgeschwindigkeit V:

$$V = \frac{M - S}{Ts} \quad .$$

Die Überschreitung Y ergibt sich aus der Zeitdauer, die dieser Anstieg auf das System wirkt:

$$Y = V * Ta = \frac{M - S}{Ts} * Ta \ .$$

Zur Verdeutlichung soll folgendes Beispiel dienen:

Die Maximaltemperatur soll 60° betragen:	M = 60°
Der gewünschte Sollwert sei 40°:	S = 40°
Die Anstiegszeit vom Sollwert zu Max.-Wert:	Ts = 100 s
Die Ausschalttotzeit sei:	Ta = 10 s

Damit ergibt sich eine Überschreitung von:

$$Y = \frac{60 - 40}{100} * 10° = 2°$$

Die Überschreitung der Differenz zwischen Sollwert und Maximalwert hängt von der Anstiegszeit und der Totzeit ab. Entsprechend ergibt sich für die Unterschreitung:

$$Y = \frac{S - U}{Tr} * Te \ .$$

U: Umgebungstemperatur
Tr: Abfallzeit (vom Sollwert zur Umgebungstemperatur bei angenähertem linearen Verlauf)
Te: Einschalttotzeit

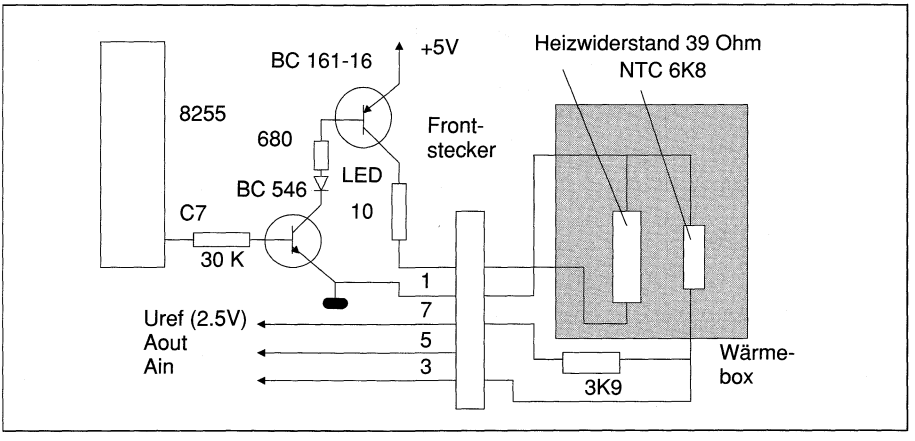


Bild 8.27: Schaltbild zur Ansteuerung einer geregelten Wärmebox

Mit der Interface-Karte läßt sich ein Zweipunktregler aufbauen. Hierzu wird auf der Platine ein Leistungsschalter aufgebaut, der eine externe Wärmebox ansteuert. Bild 8.27 zeigt den Aufbau des Systems.

Als Ansteuerausgang für die Stellgröße benutzt die Schaltung den Ausgang C7 des 8255. Als externen Heizwiderstand verwendet man beispielsweise einen Widerstand von 1/2 W und 39 Ω. Dieser erwärmt sich je nach Aufbau auf bis zu 60° oder 70° C. Die momentane Temperatur des Heizwiderstands kann man durch einen NTC (hier 6k8) erfassen. Der externe Vorwiderstand von 3k9 dient zur Vorspannungserzeugung. Nach einer geeigneten Eichung läßt sich am analogen Eingang (Pin 3 des Steckers) eine direkte Temperaturmessung der Wärmebox durchführen. Bild 8.28 zeigt einen möglichen Aufbau und dessen Anordnung auf der Platine.

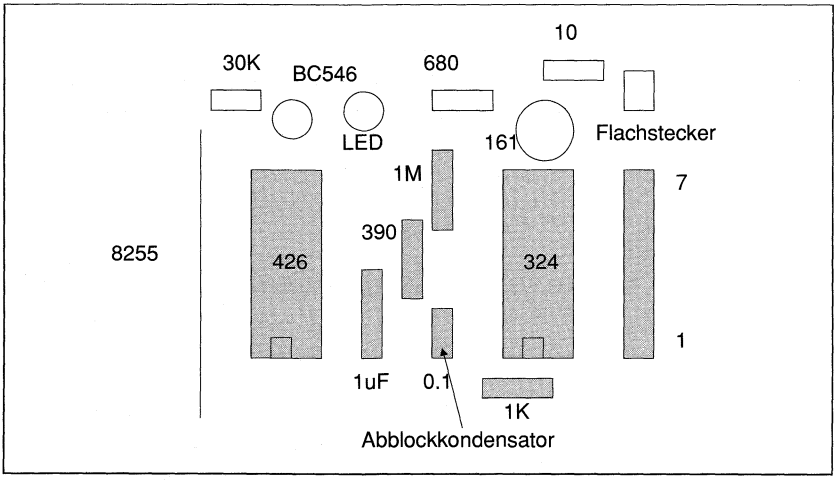


Bild 8.28: Bestückungsplan der Schaltung einer geregelten Wärmebox

Das Basic-Programm zum Betrieb des Zweipunktreglers hat folgende Form:

Programm ZPREGLER

```
REM ***********************************************
REM *Demonstrationsprogramm eines Zweipunktreglers*
REM ***********************************************
REM
REM ***********************************************
REM *                Hauptprogramm                *
REM ***********************************************
REM
```

```
REPEAT
  GOSUB prog_io
  GOSUB screen
  t=500
  REPEAT
    x$=INKEY$
    IF x$="h"
      HARDCOPY
    ENDIF
    IF x$="s"       GOSUB graph
    ENDIF
  UNTIL x$="q" OR x$=CHR$(&H1B)
UNTIL x$="q"
END
REM ***********************************************
REM *                Unterprogramme               *
REM ***********************************************
REM
PROCEDURE screen
  CLS
  DEFLINE 1,3,0,0
  BOX 5,5,635,395
  PRINT AT(50,22);" Zweipunktregler"
  DEFLINE 1,1,0,0
  BOX 20,320,600,390
  PRINT AT(4,22);" Bedienfunktionen: "
  PRINT AT(6,23);" Löschen:    ESC "
  PRINT AT(6,24);" Starten:    S "
  PRINT AT(30,23);" Hardcopy:   H "
  PRINT AT(30,24);" Pr.-Ende:   Q "
  PRINT AT(54,24);" Unterbr.: ESC"
  PRINT AT(3,4);"Ist"
  PRINT AT(3,7);"max"
  PRINT AT(3,10);"min"
  PRINT AT(3,19);"ST"
  DEFLINE 3,1,0,0
  LINE 50,150,600,150
  LINE 50,100,600,100
  DEFLINE 2,1,0,0
  LINE 50,270,600,270
  LINE 50,168,600,168
  LINE 50,65,600,65
```

```
  LINE 50,14,600,14
  DEFLINE 3,1,0,0
  y=50
  REPEAT
    LINE y,14,y,270
    y=y+50
  UNTIL y>660
RETURN
REM
PROCEDURE graph
  DEFLINE 1,1,0,0
  y=50
  REPEAT
    GOSUB suk_adc
    IF a170
      w=0
      v=1
    ENDIF
    IF a<120
      w=255
      v=11
    ENDIF
    c=w
    GOSUB out_c
    DEFLINE 1,v,0,0
    DRAW y-1,290 TO y,290
    DEFLINE 1,1,0,0
    PLOT y,270-a     y=y+1
    PAUSE (t/20-1)
  UNTIL y=600 OR INKEY$=CHR$(&H1B)
RETURN
REM
PROCEDURE suk_adc
  r=128
  z=8
  a=0
  REPEAT
    a=a+r
    GOSUB out_a
    GOSUB read_c
    IF ODD(c)
      a=a-r
```

```
    ENDIF
    z=z-1
    r=r/2
  UNTIL z=0
RETURN
REM
PROCEDURE prog_io
  x=PEEK(16385798)
  x=PEEK(16384512)
  x=PEEK(16415232)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE out_a
  x=PEEK(16385536+2*a)
  x=PEEK(16390656)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE out_c
  x=PEEK(16385536+2*c)
  x=PEEK(16407040)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE read_c
  c=DPEEK(16404992)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE print_out
  GOSUB read_c
  IF c=65523
    y=1
  ENDIF
  IF c=65535
    y=2
  ENDIF
  GOSUB out_a
  c=127
  GOSUB out_c
  c=255
```

```
  GOSUB out_c
RETURN
```

Wie der Ausdruck bereits zeigt, sind zahlreiche Teile des Programms aus dem analogen Speicheroszilloskop übernommen worden. In der grafischen Darstellung findet man jedoch noch zusätzlich die Darstellung des momentanen Ist-Wertes mit den Begrenzungen für den maximalen und den minimalen Wert. Entsprechend arbeitet der Zweipunktregler streng genommen auch nicht mit einer vorgegebenen Totzeit; vielmehr besitzt er eine feste Hysterese, zwischen deren Eckpunkten der Regelvorgang abläuft. Unter der grafischen Darstellung wird jeweils auch der Zustand des Stellglieds angezeigt, wobei ein dicker Balken den "Ein"-Zustand signalisiert. Bild 8.29 zeigt einen typischen Regelablauf. Die Anstiegs- und die Abfallzeit zwischen den Eckpunkten des Reglers hängen stark von dem Aufbau der Wärmebox ab. Eine gute Isolation führt zu langen Abfallzeiten, während eine kleine Wärmekapazität des Quarzes und des NTC-Widerstands zu einer kurzen Anstiegszeit beitragen.

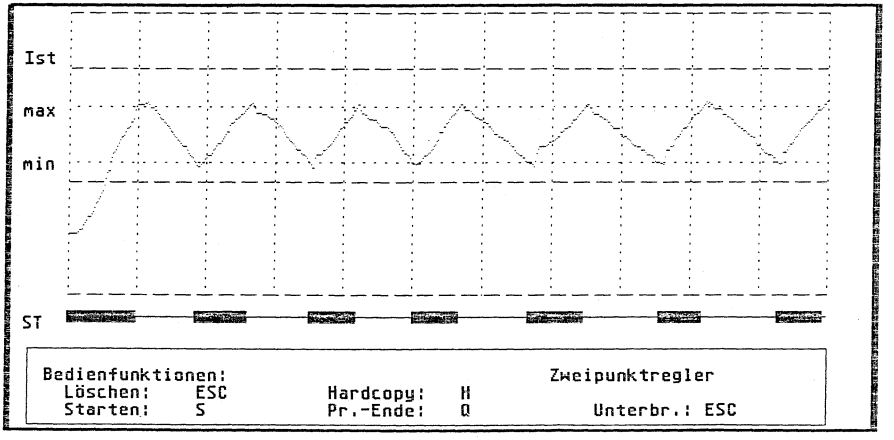


Bild 8.29: Hardcopy des Regelverlaufs bei einem Zweipunktregler (Wärmebox)

8.17 Inbetriebnahme des seriellen Interfaces

Zum Betrieb des seriellen Interfaces wurde der wesentliche Baustein in Form des 8251 bereits bestückt. Eingangsseitig sind alle notwendigen Signale auf der Platine verdrahtet. Die Kommunikation läuft dabei ganz ähnlich ab, wie das bereits mit dem Baustein 8255 für parallele Interface-Aufgaben erfolgreich durchgeführt wurde.

Ein direkter Betrieb des Bausteins ohne einen weiteren individuellen Aufbau bietet jedoch recht wenige Möglichkeiten, so daß man am einfachsten direkt mit einem konkreten Aufbau beginnt.

Für alle asynchronen Applikationen (und das ist die Mehrzahl) benötigt der Baustein mehrere Takt-Signale, die sowohl den internen als auch den externen Ablauf vorgeben. So braucht man als erstes ein Clock-Signal für den internen Ablauf des Bausteins. Dieses Signal muß TTL-Pegel haben und über eine Frequenz verfügen, die mindestens den 30fachen Wert der Baud-Rate aufweist. Sinnvoll sind hier alle Frequenzen von 300 kHz bis 3 MHz. Zusätzlich erhält der Baustein noch das eigentliche Baud-Raten-Signal, das erst durch die interne Teilung zur gewünschten Baud-Rate führt. Da der 8251 als Teilungsfaktoren die Werte 16 und 64 zuläßt, benötigt man als maximale Frequenz entweder 153,6 kHz oder 614,4 kHz für die Baud-Rate von 9600. Das letzte Signal zur Baud-Raten-Voreinstellung sollte alle Abstufungen zulassen, die auch zu den Baud-Raten 4800, 2400, 1200, 600, 300 und 110 führen.

Es gibt zweifellos unzählige Möglichkeiten, wie man Oszillatoren aufbaut, die diesen Anforderungen genügen. Bild 8.30 zeigt eine Applikation, die mit recht wenigen Bauteilen alle notwendigen Frequenzen erzeugt.

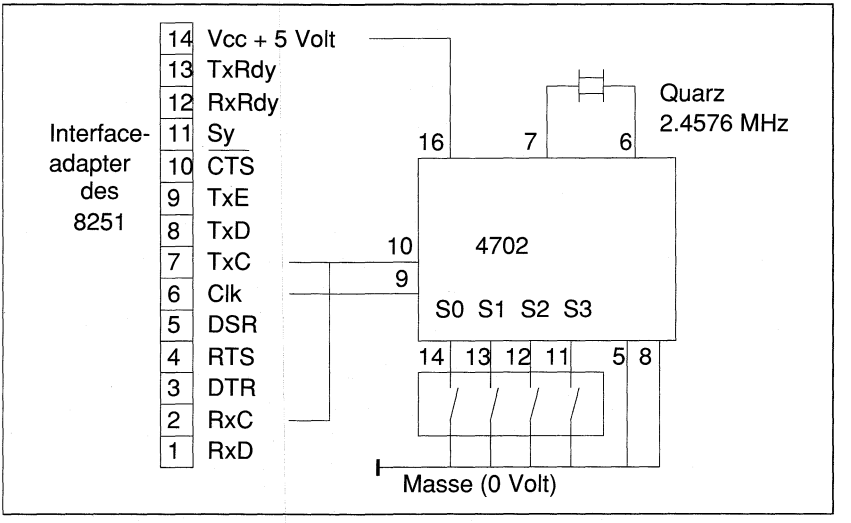


Bild 8.30: Beschaltung des 8251 und des 4702

Kernstück der Anwendung ist der Baud-Raten-Generator 4702 (Fairchild), der nur noch einen Quarz und einen Schiebeschalter (zur Voreinstellung) zum Betrieb braucht.

Der Baustein erhält seine hochstabile Grundfrequenz durch einen Quarz von 2,4576 MHz. Der interne Oszillator stellt diese Grundfrequenz am Clock-Output (Pin 9) direkt für den 8251 zur Verfügung. Die Baud-Raten lassen sich am Ausgang (Pin 10) abgreifen, wobei der interne Multiplexer durch einen vierfachen Schiebeschalter selektiert werden muß.

Bei einem internen Teilerverhältnis von 16 ergeben sich je nach Schalterstellung folgende Baud-Raten für den seriellen Baustein 8251:

S3	S2	S1	S0	Baud-Rate
0	0	0	0	nicht erlaubt
0	0	0	1	nicht erlaubt
0	0	1	0	50
0	0	1	1	75
0	1	0	0	134,5
0	1	0	1	200
0	1	1	0	600
0	1	1	1	2400
1	0	0	0	9600
1	0	0	1	4800
1	0	1	0	1800
1	0	1	1	1200
1	1	0	0	2400
1	1	0	1	300
1	1	1	0	150
1	1	1	1	110

Da der Baustein interne Widerstände besitzt, die seine Eingänge bei Nichtbeschalten hochziehen, braucht der Schiebeschalter nur einseitig mit Masse verbunden werden. Der Zustand "1" ergibt sich dann, wenn sich der Schalter in der Stellung "Off" (offen) befindet.

Da man selten einen Unterschied zwischen der Ausgabe-Baud-Rate und der Eingabe-Baud-Rate macht, werden die beiden Takteingänge RxC und TxC des 8251 miteinander verbunden und an den Pin 10 des 4702 angeschlossen. Ein zusätzlicher Abblockkondensator sorgt für eine hinreichende Aussiebung der Betriebsspannung in der Nähe des 4702.

Es ist an dieser Stelle empfehlenswert, alle Frequenzsignale über ein Oszilloskop zu kontrollieren. Eventuelle Fehlschaltungen oder Kurzschlüsse lassen sich leichter aufspüren.

In allen nachfolgenden Applikationen benutzt man ausschließlich die Baud-Rate von 9600. Hierzu muß der Schiebeschalter in die Stellung 1 0 0 0 gebracht werden.

Mit diesem Aufbau ist der 8251 bereits für alle seriellen Anwendungen hinreichend gerüstet. Da er aber ein NMOS- oder CMOS-Baustein ist, erzeugt er lediglich Amplituden im TTL-Bereich. Zum Anschluß eines RS 232 C-Terminals (V.24) braucht man jedoch symmetrische Signale, die Pegel von –12 bis +12 Volt erreichen. Da dem Atari ST jegliche Spannungswerte in diesem Bereich fehlen, muß man sich zusätzliche Bauteile leisten, die aus den vorhandenen 5 Volt die gewünschten +/– 12 Volt erzeugen.

Besonders einfach gelingt die direkte Umsetzung der TTL-Signale des 8251 in die Standard V.24-Signale durch den Interface-Baustein MAX 232 (Fa. Maxim). Dieser Baustein konvertiert nicht nur die Signale des 8251, sondern er beinhaltet auch eine interne Spannungsquelle, die aus den vorhandenen 5 Volt die gewünschten +/– 12 Volt erzeugt. Zum Betrieb sind lediglich vier weitere Kondensatoren notwendig, die dem internen Oszillator und die Stabilisierung der neuen Spannungen dienen. Die Schaltung ist im Bild 8.31 ersichtlich.

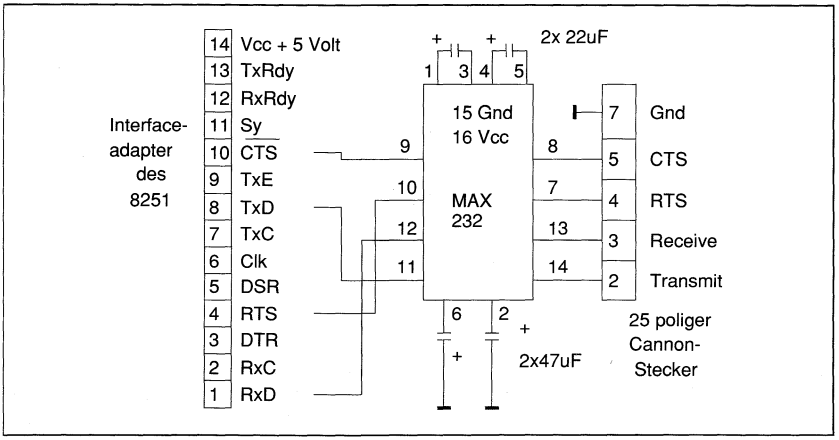


Bild 8.31: Beschaltung des 8251, des MAX 232 und des 25poligen Steckers

Der Baustein kann ohne weitere Komponenten direkt mit dem RS-232-C-Stecker verbunden werden. Da er kurzschlußfest ist, sind keinerlei Zusatzmaßnahmen notwendig. Für den eigenen Weiterbau der Platine besteht sogar die Möglichkeit, seine Referenzspannungen von ca. +/– 10 Volt anzuzapfen. Dabei sollte die Belastung jedoch nur wenige Mikroampere betragen. Bild 8.32 zeigt die Anordnung der Bauteile als Beispiel.

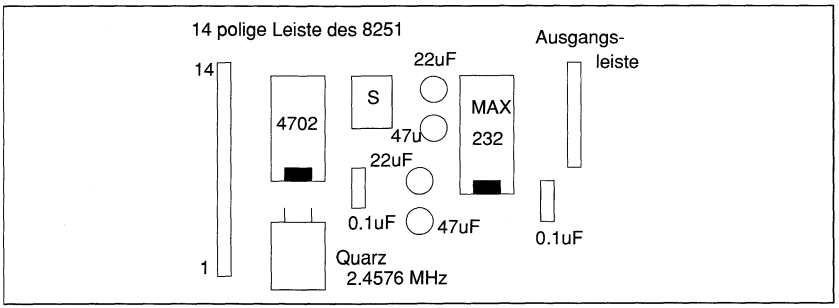


Bild 8.32: Bestückungsplan der Schaltung für die serielle Datenübertragung

8.18 Serielle Dateneingabe

Das erste Programm für den 8251 mit den im vorigen Abschnitt beschriebenen Bausteinen benutzt das neue System zur seriellen Eingabe von Daten. Wie auch bei allen Applikationen des parallelen Bausteins 8255 wird die Adressvorwahl zur Dateneingabe verwendet. Dabei ergibt sich folgende Funktionstabelle für den 8251:

```
Adresse    A14 A13 A12 A11 A10 A9     A8-A1

Funktion

Schreiben          1   1
Lesen              1   0
Reset              0   1
Select (CS)                1   0
Latch                      1   1

               A0                     Daten  (des 8251)
```

Genau wie bei allen Applikationen des 8255 verwendet man die Basisadresse FA0000. Zu ihr werden danach alle notwendigen Adreßkombinationen addiert, die der Funktion und dem Datenwort entsprechen. Ein Beispiel soll diese Vorgehensweise genauer darstellen:

Möchte man z.B. das Datum 37Hex in den Kontrollport des 8251 laden, so braucht man insgesamt drei Operationen. Bei der ersten wird der Wert 37Hex im Latch abgelegt. Die notwendige Bitkodierung hat dabei folgende Form:

```
0 0 0  0 1 1 0  0 1 1 0  1 1 1 0
```

Die ersten drei Nullen entfallen auf die Adressen A14 bis A12. Sie verbleiben auf 0, da keine Schreib- oder Lesefunktion gefordert wird. Die nächsten 4 Bit haben die Form 0 1 1 0 und entfallen auf den Adreßbereich A11 bis A8. Mit ihnen wird die Funktion "Latch" angesprochen. Zusätzlich folgt die Information 37Hex, bei der zu berücksichtigen ist, daß A0 stets auf 0 bleibt. Die Information 37Hex erscheint damit um eine Stelle nach links verschoben, so daß A7 in das Bit von A8 übergeht.

Nach der ersten Information muß nun eine Schreibfunktion auf den Kontrollport des 8251 erfolgen. Dazu wird die Schreib-Kodierung und das Selektiersignal (CS) angesprochen:

```
0 1 1  1 1 0 0  0 0 0 0  0 0 0 0
```

Abschließend schaltet man alle Signale wieder aus, so daß der Baustein im Ruhezustand verweilt (Kodierung 000 0000 0000 0000).

Die gesamte Sequenz hat damit im Bezug auf die Basisadresse folgendes Aussehen:

FA066E	entspr. 16385646	37Hex auf Latch
FA3C00	entspr. 16399360 WR	CS mit A0=1
FA0000	entspr. 16384000	alles wieder aufheben

Der 8251 benötigt im Gegensatz zum 8255 insgesamt zwei Programmierworte. Da er jedoch nur über einen Kontrollport verfügt, kommt es auf die richtige Sequenz dieser Worte an. Normalerweise bringt das Reset-Signal eine eindeutige Abfolge, so daß eine richtige Identifizierung zwischen dem ersten und dem zweiten Byte ermöglicht ist. Da man aber nur selten weiß, ob der Baustein ein Reset-Signal erhalten hatte oder bereits durch ein anderes Programm programmiert wurde, läßt sich diese Strategie nicht optimal einhalten. Besser verläuft daher eine Programmierung, die unabhängig vom Reset-Zustand arbeitet. Dazu wird der 8251 zuerst dreimal mit 00 geladen. Danach erhält er einen Soft-Reset durch die Information 40Hex, worauf die eigentliche Programmierung ohne weitere Schwierigkeiten erfolgen kann. In den nächsten Beispielen wird diese Taktik angewandt, wobei der Baustein danach mit dem Teilungsfaktor 16, mit einem Stop-Bit und mit acht Daten-Bits programmiert wird.

Das erste Basic-Programm dient der einfachen Dateneingabe. Es können dabei alle ASCII-Zeichen empfangen werden, die dann am Bildschirm untereinander erscheinen:

Programm SER_IN

```
REM ***********************************************
REM *      Programm zur seriellen Eingabe         *
REM ***********************************************
REM
GOSUB prog_ser
CLS
q=0
REPEAT
  GOSUB input
  y$=CHR$(c)
  PRINT y$
UNTIL q=1
END
REM
PROCEDURE prog_ser
  REM Aufruf folgender Funktionen Newprog, Latch mit 40H, 4EH, 37H
  REM   danach jeweils CS und Write
  y=3
  REPEAT
    x=PEEK(16385536)
    x=PEEK(16399360)
```

```
    x=PEEK(16384000)
    y=y-1
  UNTIL y=0
  x=PEEK(16385664)
  x=PEEK(16399360)
  x=PEEK(16384000)
  x=PEEK(16385692)
  x=PEEK(16399360)
  x=PEEK(16384000)
  x=PEEK(16385646)
  x=PEEK(16399360)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE get_stat
  REM Aufruf des Status mit A0=1, CS und Read
  s=DPEEK(16397312)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE char_out
  REM Aufruf der Ausgabe mit Latch, CS und Write
  x=PEEK(16385536+2*c)
  x=PEEK(16391168)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE char_in
  REM Aufruf der Eingabe mit A0=0, CS und Read
  c=DPEEK(16389120)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE output
  REPEAT
    GOSUB get_stat
  UNTIL ODD(s)
  GOSUB char_out
RETURN
REM
PROCEDURE input
  REPEAT
```

```
    GOSUB get_stat
    IF INKEY$=CHR$(&H1B)
      q=1
    ENDIF
  UNTIL ODD(INT(s/2)) OR q=1
  GOSUB char_in
RETURN
```

Zur Prüfung des Programms kann jedes Terminal oder jeder Rechner verwendet werden, der serielle Daten sendet. Je nach Vorwahl des Schiebeschalters lassen sich auch andere Baud-Raten als 9600 verwenden. Die Datenbreite und die Anzahl der Stop-Bits können in der Programmierung vereinbart werden.

8.19 Serielle Ausgabe und Terminal-Betrieb

Die im vorigen Abschnitt beschriebene Interface-Schaltung erlaubt den direkten Full-Duplex-Verkehr, bei dem gleichzeitig Daten gesendet und empfangen werden können. Die serielle Ausgabe und der Terminal-Betrieb sind daher nur noch eine "Formsache". Zum Betrieb einer Datenausgabe benötigt man nicht unbedingt einen zweiten Atari ST oder ein Terminal, vielmehr lassen sich auch Drucker oder ähnliche serielle Interface-Geräte anschließen.

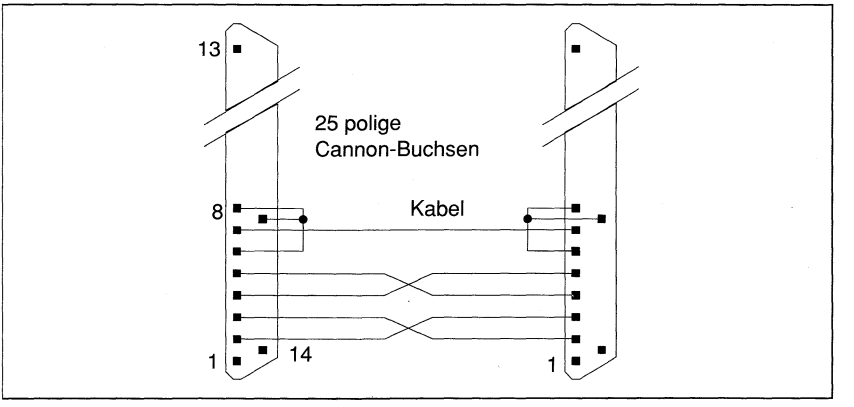


Bild 8.33: Belegung der 25poligen Buchsen für die serielle Datenübertragung

Eine einwandfreie Funktion wird jedoch nur dann gewährleistet, wenn neben den Datenleitungen (RxD und TxD) auch die Kontrolleitungen (CTS und RTS) richtige Pegel aufweisen. Der Baustein 8251 kann nur dann Daten senden, wenn die Kontrolleitung CTS eine

Spannung von mehr als 3 Volt (typisch 12 Volt am Stecker) aufweist. Bei allen Dateneinrichtungen, die diesen Pegel nicht zur Verfügung stellen, sollten die Signale CTS und RTS (Pins 4 und 5) am seriellen Interface-Stecker des Atari verbunden werden. Sofern sich jedoch der gegenüberliegende Teilnehmer normgerecht verhält, braucht man zum Datenverkehr lediglich ein Interface-Kabel, bei dem die Leitungen 2 und 3 sowie 4 und 5 kreuzweise miteinander verbunden sind (Bild 8.33). Die Pins 6,8 und 20 sind am Teilnehmer selbst gebrückt. Die Masseleitung wird durch den Anschluß des Pins 7 beidseitig übertragen.

Das Ausgabeprogramm erzeugt eine ASCII-Zeile mit allen Zeichen vom "Space" bis zum "Z" in aufsteigender Reihenfolge. Es hat folgende Form:

Programm SER_OUT

```
REM ***********************************************
REM *      Programm zur seriellen Ausgabe         *
REM ***********************************************
REM
GOSUB prog_ser
c=32
REPEAT
  GOSUB output
  c=c+1
UNTIL c=91
c=13
GOSUB output
c=10
GOSUB output
END
REM
PROCEDURE prog_ser
  REM Aufruf folgender Funktionen Newprog, Latch mit 40H, 4EH, 37H
  REM   danach jeweils CS und Write
  y=3
  REPEAT
    x=PEEK(16385536)
    x=PEEK(16399360)
    x=PEEK(16384000)
    y=y-1
  UNTIL y=0
  x=PEEK(16385664)
  x=PEEK(16399360)
  x=PEEK(16384000)
  x=PEEK(16385692)
  x=PEEK(16399360)
```

```
  x=PEEK(16384000)
  x=PEEK(16385646)
  x=PEEK(16399360)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE get_stat
  REM Aufruf des Status mit A0=1, CS und Read
  s=DPEEK(16397312)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE char_out
  REM Aufruf der Ausgabe mit Latch, CS und Write
  x=PEEK(16385536+2*c)
  x=PEEK(16391168)
  x=PEEK(16384000) RETURN
REM
PROCEDURE char_in
  REM Aufruf der Eingabe mit A0=0, CS und Read
  c=DPEEK(16389120)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE output
  REPEAT
    GOSUB get_stat
  UNTIL ODD(s)
  GOSUB char_out
RETURN
```

Genau wie das Eingabeprogramm greift das Programm SER_OUT auf Prozeduren zurück, die bei der Weiterverarbeitung eigener Programmideen recht nützlich sind. Die Routinen orientieren sich an der Funktionsweise des Bausteins 8251 und stellen komplette Einheiten dar, die sich unabhängig in andere Programmteile einbauen lassen. Zu den wichtigen Prozeduren gehören: OUTPUT, INPUT, GET_STAT, CHAR_OUT, CHAR_IN und PROG_SER. Die Namen der Routinen sprechen für sich.

Ein weiteres Interface-Programm zur seriellen Kommunikation ist das Programm TERMINAL. Es hat folgende Form:

Programm TERMINAL

```
REM ********************************************
REM *              Terminalprogramm              *
REM ********************************************
REM
GOSUB prog_ser
CLS
q=0
REPEAT
  y$=INKEY$
  IF y$CHR$(0)
    c=ASC(y$)
    GOSUB output
  ENDIF
  IF y$=CHR$(&H1B)
    q=1
  ENDIF
  GOSUB get_stat
  IF ODD(INT(s/2))
    GOSUB char_in
    y$=CHR$(c-65280)
    PRINT y$;
  ENDIF
UNTIL q=1
END
REM
PROCEDURE prog_ser
  REM Aufruf folgender Funktionen Newprog, Latch mit 40H, 4EH, 37H
  REM   danach jeweils CS und Write
  y=3
  REPEAT     x=PEEK(16385536)
    x=PEEK(16399360)
    x=PEEK(16384000)
    y=y-1
  UNTIL y=0
  x=PEEK(16385664)
  x=PEEK(16399360)
  x=PEEK(16384000)
  x=PEEK(16385692)
  x=PEEK(16399360)
  x=PEEK(16384000)
  x=PEEK(16385646)
  x=PEEK(16399360)
  x=PEEK(16384000)
RETURN
```

```
REM
PROCEDURE get_stat
  REM Aufruf des Status mit A0=1, CS und Read
  s=DPEEK(16397312)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE char_out
  REM Aufruf der Ausgabe mit Latch, CS und Write
  x=PEEK(16385536+2*c)
  x=PEEK(16391168)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE char_in
  REM Aufruf der Eingabe mit A0=0, CS und Read
  c=DPEEK(16389120)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE output
  REPEAT
    GOSUB get_stat
  UNTIL ODD(s)
  GOSUB char_out
RETURN
```

Das Programm verwendet die Interface-Karte als serielles Terminal. Dabei werden die Standardfunktionen des VT-52 übernommen. Die Arbeitsweise ist damit identisch zu der bereits im Atari befindlichen VT-52 Emulation über die serielle Schnittstelle. Im Gegensatz zu dieser lassen sich jedoch auch weitere Baud-Raten erreichen und Datenformate einstellen, die sonst mit dem seriellen Interface nicht möglich sind.

8.20 Einstieg in lokale Netze

Die Funktion der Interface-Karte ermöglicht prinzipiell bereits den Aufbau eines lokalen Netzes. Dabei ist die einfachste Topologie der Ringverbindung direkt ohne weitere Bauteile erreichbar. Man benutzt dabei jeweils den Ausgang des MAX 232 und verbindet diesen mit dem Eingang des nächsten Teilnehmers. Bild 8.34 zeigt einen Verbund von vier Rechnern mit einer Ringverbindung, die aus vier Punkt-zu-Punkt Verbindungen besteht. Die interne Software muß lediglich dafür sorgen, daß jede Information vom Eingang gelesen wird und danach sofort am Ausgang erscheint. Dadurch kann jeder angeschlossene Teilnehmer mit einem Versatz von einem Byte die Gesamtinformation auslesen. Lediglich am Sender bleibt

der Eingang unberücksichtigt, da dieser als Verursacher des Informationsflusses die Daten verwirft.

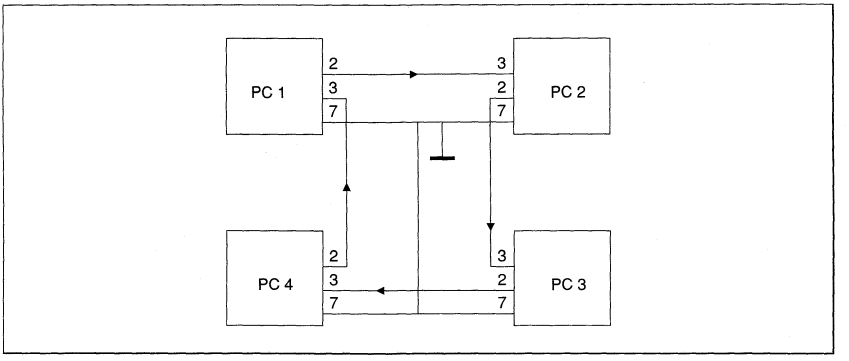


Bild 8.34: Aufbau eines lokalen Netzes aus vier Atari ST

Zur Zugriffsregelung erhält jeder Teilnehmer vor dem Beginn der Transportkette eine eigene Adresse, die ihn von allen anderen unterscheidet. Danach sendet der erste Teilnehmer eine Information aus, die jeder weitere Teilnehmer nur vom internen Empfänger zum Sender durchreicht. Diese »Durchreichefunktion« bleibt so lange erhalten, bis ein Teilnehmer nach einer speziellen »Achtung-Kodierung« seine eigene Adresse erkannt hat und einen vollständigen Datenblock abgespeichert hat. Danach wird er selbst zum nächsten Sender.

Ein einfacher Datenverkehr kann dadurch erfolgen, daß man als Zeichen nur Ziffern zuläßt und als »Achtung-Zeichen« den »*« vereinbart. Ein Datensatz kann damit folgende Form haben:

```
*  ADR  D1 D2 D3 D4 D5 D6 D7 D8 Summe

*     :    Achtung-Zeichen
ADR   :    Adreßbyte
D1-D8:     Datenbytes
Summe:     Modulo-8 Summe über Adresse und Daten
```

Damit muß das verwendete Interface-Programm alle ankommenden Zeichen so lange weitergeben, bis ein »*« erscheint. Nach diesem ASCII-Zeichen folgt die Adresse, die jeder Empfänger mit seiner eigenen vergleicht. Bei fehlerhafter Adresse (also der Teilnehmer ist nicht gemeint) verwirft er den angefangenen Datensatz und setzt die Weitergabe lückenlos fort. Sofern die Adresse aber stimmt, legt der richtige Teilnehmer eine interne Datei an, die alle weiteren acht Folge-Daten ablegt. Am Ende wird die Summe über die Adresse und die Daten berechnet und mit der Übertragungssumme verglichen. Einfache Übertragungsfehler lassen sich damit bereits sehr effektiv aufspüren.

Ein einfaches Empfangsprogramm hat die Bezeichung LAN und ist im folgenden dargestellt:

Programm LAN

```
REM ***********************************************
REM *  Demonstrationsprogramm eines Einfach-LANs   *
REM ***********************************************
REM
GOSUB prog_ser
DIM b(11)
i=0
REPEAT
  GOSUB input
  GOSUB output
  IF c=ASC("*")
    i=1
  ENDIF   IF i=2
    IF cASC("5")
      i=0
    ENDIF
  ENDIF
  IF i0
    b(i-1)=c
    i=i+1
  ENDIF
UNTIL i=12 OR INKEY$=CHR$(&H1B)
i=1
k=0
REPEAT
  k=k+b(i)
  i=i+1
UNTIL i=12
PRINT " Datensatz abgelegt "
i=1
PRINT
PRINT " ";
REPEAT
  PRINT CHR$(b(i));
  i=i+1
UNTIL i=12
PRINT
PRINT "Summe: (berechnet) ",k
END
```

```
REM
PROCEDURE prog_ser
  REM Aufruf folgender Funktionen Newprog, Latch mit 40H, 4EH, 37H
  REM   danach jeweils CS und Write
  y=3
  REPEAT
    x=PEEK(16385536)
    x=PEEK(16399360)
    x=PEEK(16384000)
    y=y-1
  UNTIL y=0
  x=PEEK(16385664)
  x=PEEK(16399360)
  x=PEEK(16384000)
  x=PEEK(16385692)
  x=PEEK(16399360)
  x=PEEK(16384000)
  x=PEEK(16385646)
  x=PEEK(16399360)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE get_stat
  REM Aufruf des Status mit A0=1, CS und Read
  s=DPEEK(16397312)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE char_out
  REM Aufruf der Ausgabe mit Latch, CS und Write
  x=PEEK(16385536+2*c)
  x=PEEK(16391168)
  x=PEEK(16384000)
RETURN REM
PROCEDURE char_in
  REM Aufruf der Eingabe mit A0=0, CS und Read
  c=DPEEK(16389120)
  x=PEEK(16384000)
RETURN
REM
PROCEDURE output
  REPEAT
```

```
    GOSUB get_stat
  UNTIL ODD(s)
  GOSUB char_out
RETURN
REM
PROCEDURE input
  REPEAT
    GOSUB get_stat
  UNTIL ODD(INT(s/2))
  GOSUB char_in
  c=c-65280
RETURN
```

Man erkennt, daß dieses Programm alle Eingabezeichen sofort an die Ausgabe weiterleitet. Dadurch sind alle Ringteilnehmer gleichberechtigt. Sie erhalten den gesamten Datensatz jeweils mit einem Versatz von einem Zeichen. Der Sender selbst benutzt ein Sendeprogramm und verwirft damit alle Daten am Eingang.

Nachdem das »*« erkannt wurde, wartet das Programm auf das nächste Byte, das wie vereinbart die Adresse darstellt. In dem Interface-Programm wurde die Adresse »5« gewählt. Bei Adreßgleichstand eröffnet das Basic-Programm eine Datei, die alle nachfolgenden neun Zeichen (also acht Daten und eine Summe) abspeichert. Nach Beendigung der Dateieingabe berechnet das Basic-Programm die Quersumme und gibt den gesamten Datensatz mit der Summe auf dem Bildschirm aus.

Die so erzeugte LAN-Verbindung läßt sich bis zu zehn Teilnehmern betreiben, wobei der Maximalabstand 15 Meter nicht überschritten werden darf.

Das dargestellte Programm LAN ist eher als exemplarisch anzusehen und kann vom Benutzer beliebig erweitert werden. Komplette Programme zum Datenverkehr innerhalb lokaler Netze würden den Umfang dieses Buches sprengen. Alleine eine Abdeckung der zweiten Schicht des ISO-Modells benötigt ein Interface-Programm, das den doppelten Umfang dieses Buches hätte. Dennoch lassen sich einfache LAN-Aufgaben durch ähnliche Programme durchführen wie das hier vorgestellte. Die Funktion und die Erscheinungsform hängt jedoch stark von der gewünschten Applikation ab.

Verbindungen mit der RS 232 C haben den Nachteil, daß sie noch recht störanfällig sind und darüber hinaus keinen Potentialausgleich zulassen. Möchte man daher zwei (oder mehrere) Atari ST verbinden, die weiter als 15 Meter entfernt sind und die eventuell auch noch durch verschiedene Netze betrieben werden, so versagt die Schnittstelle oftmals.

Eine interessante Alternative bietet hier die Lichtwellenleiterverbindung, bei der man das elektrische Drahtmedium durch ein optisches ersetzt. Kunststofflichtwellenleiter liegen heute in einer niedrigen Preisklasse, so daß sich derartige Verbindungen recht preiswert realisieren lassen. Zur Handhabung des Plastik-Lichtwellenleiters benötigt man nur noch ein

scharfes Messer mit etwas Schmirgelpapier. Interfaceelemente werden von zahlreichen Firmen angeboten (z.B. Hirschmann oder HP). Die Interface-Karte erlaubt es, direkt am 8251 derartige LWL-Koppler zu betreiben. Je nach Wahl des Herstellers und der verwendeten Technik differieren die Adaptionsverfahren, so daß an dieser Stelle nur auf die technischen Beschreibungen der Hersteller verwiesen sein kann. Bild 8.35 zeigt derartige Interface-Elemente mit den Steckern und dem Plastikkabel der Firma HP.

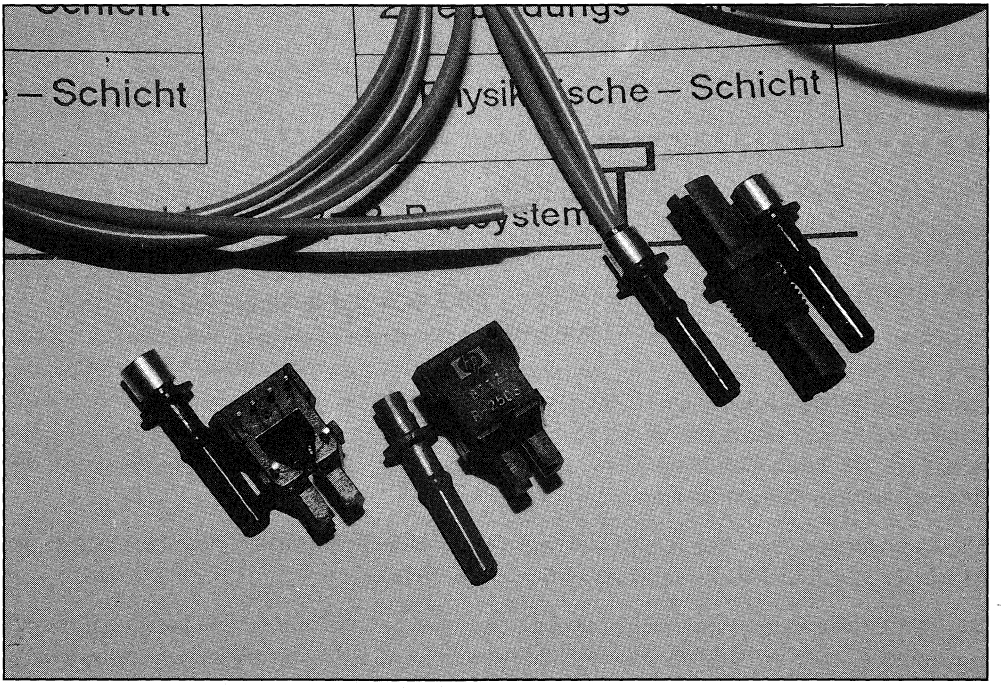


Bild 8.35: Bauteile zum Aufbau eines Lichtwellenleiters (LWL)

LWL-Verbindungen mit den abgebildeten Interface-Bauteilen garantieren einen einwandfreien Datenverkehr bis zu 300 Meter bei einer Baud-Rate von 9600. Wegen der fehlenden Drahtverbindung sind auch Potentialbarrieren bis in den Kilovolt-Bereich ohne Schwierigkeiten zu überwinden.

9 Anhang

9.1 Fehlerbehebung auf der Interface-Platine

Dieser Abschnitt gibt Ihnen einige Informationen, falls die Interface-Platine nicht in der gewünschten Art und Weise funktioniert.

Zur eventuellen Fehlerbeseitigung sind ein Voltmeter und ein Ohmmeter dienlich. Das Voltmeter sollte einen Eingangswiderstand haben, der größer als 20 kOhm ist und Spannungen im Bereich von 0–5 Volt gut erfaßt. Das Ohmmeter sollte im Widerstandsbereich bis ca. 100 kOhm messen können. Für die Messung der Stromaufnahme wird zusätzlich noch ein Netzgerät mit einer Ausgangsspannung von 5 Volt (min. 0,5 Ampere) und ein Strommesser benötigt. Für alle Messungen sind kleine Digitalvoltmeter besonders geeignet, die alle notwenigen Meßbereiche abdecken.

Vor den meßtechnischen Untersuchungen sollten Sie eine visuelle Kontrolle der Platine durchführen, da in fast allen Fällen Kurzschlüsse oder fehlerhafte Lötungen als Ursache von Fehlfunktionen auftreten. Auch die Wahrscheinlichkeit, daß ein Bauteil nicht einwandfrei arbeitet, ist überaus gering. Sie sollten deshalb erst alle anderen Möglichkeiten untersuchen, bevor Sie ein Bauteil (Widerstand, Kondensator oder IC) austauschen.

Fehlerbehandlung A

Die Platine enthält in diesem Fall noch keinerlei aktive Elemente. Alle Verbindungen zu der Steckerleiste führen noch zu den leeren Verbindungen der IC-Sockel. Eine Beeinträchtigung der Funktionsweise kann höchstens durch Kurzschlüsse oder falsche Polarität der Kondensatoren hervorgerufen worden sein.

Untersuchen Sie die Platine ganz genau, ob benachbarte Lötaugen nicht durch eine Unsauberkeit miteinander verbunden wurden. Mittels des Ohmmeters können Sie eventuelle Kurzschlüsse auch messen (Ohmbereich 100-200 kOhm). Alle Steckverbindungen müssen hochohmig gegeneinander sein.

Sollten Sie ein Netzgerät zur Verfügung haben, so können Sie am Kondensator C1 (22 uF) 5 Volt anlegen. Die 0 Volt (Masse) wird am oberen und die 5 Volt am unteren Ende angelegt (Platine von Bestückungsseite gesehen). Aus dem Netzgerät darf in diesem Fall kein Strom

gezogen werden. Alle Steckerzungen sind bei der Messung der Spannung potentialfrei (sie zeigen 0 Volt an); lediglich die Versorgungszungen 1,2 und 39,40 führen 5 Volt.

Fehlerbehandlung B

Ein eventueller Fehler nach dem Bestücken der integrierten Schaltkreise kann eine falsche Positionierung der ICs als Ursache haben. Untersuchen Sie daher bitte, ob alle Schaltkreise die richtige Position und die vorgesehene Orientierung aufweisen. Überprüfen Sie gegebenenfalls auch die Beschriftung der Schaltkreise.

Möglicherweise macht sich ein Kurzschluß bei einer internen Leiterbahn erst jetzt bemerkbar. Mit einem Netzgerät kann nun der Stromverbrauch (in Reihe zu einem Amperemeter) gemessen werden. Er sollte 100 mA nicht überschreiten.

Fehlerbehandlung C

Wenn erst nach dem Bestücken der beiden Peripheriebausteine ein Fehler auftritt, so kann dieses Verhalten nur durch ein falsches Einsetzen der Bausteine zustande kommen.

Überprüfen Sie die Position und die Orientierung der Bausteine. Entfernen Sie diese gegebenenfalls erneut und betreiben die Interface-Platine ohne die Bausteine. Bestücken Sie dann die Bausteine erneut und führen Sie nach jedem Einfügen eine Funktionskontrolle durch.

Fehlerbehandlung D

Falls die Leuchtdiode nicht in der angegebenen Art und Weise ein- und ausschaltet oder blinkt, so kann das mehrere Fehlerursachen haben. Die einfachste Ursache besteht darin, daß die Leuchtdiode in falscher Polarität bestückt wurde. Ein Ohmmeter oder ein Netzgerät mit vorgeschaltetem Widerstand klärt diesen Sachverhalt sofort auf.

Viel häufiger ist jedoch ein Programmierfehler als Ursache aufzuführen. Überprüfen Sie daher das Programm. Jede Zeile muß genau stimmen und auch die richtigen Werte enthalten. Bereits eine fehlerhafte oder unterlassene Zahl führt zur Fehlfunktion.

Manchmal machen sich auch erst jetzt Kurzschlüsse oder offene Lötstellen bemerkbar, die noch nicht gefunden wurden. Es ist daher ratsam, alle Maßnahmen von A bis C erneut durchzuführen und damit eine mögliche Ursache zu finden.

9.2 Farbcode-Tabelle

Die folgende Tabelle richtet sich nach der IEC-Publikation 62. Nach dieser Tabelle kann der Widerstandswert und die Toleranz an den Farbringen abgelesen werden. Dazu wird der Widerstand so gehalten, daß der Gold- oder Silberring als letzter auf der rechten Seite liegt.

Farbe	**1. Ring**	**2. Ring**	**3. Ring**
schwarz	0	0	.0
braun	1	1	0
rot	2	2	00
orange	3	3	000
gelb	4	4	0000
grün	5	5	00000
blau	6	6	000000
violett	7	7	
grau	8	8	
weiß	9	9	

Der Silberring zeigt eine zehn- und der Goldring eine fünfprozentige Toleranz an.

Beispiele:

blau	grau	braun	Gold	680 Ohm	(5%)
braun	schwarz	rot	Gold	1000 Ohm	(1 kOhm)
braun	schwarz	orange	Gold	10000 Ohm	(10 kOhm)
rot	rot	orange	Silber	22 kOhm	(10%)

9.3 Der Befehlssatz des Prozessors MC 68000

Liste der mnemonischen Befehle des 68000-Assemblers

ABCD	Add Decimal with Extend, dezimale (BCD) Addition
ADD	Add Binary, binäre Addition
ADDA	Add Address, binäre Addition mit Adreßregister als Ziel
ADDI	Add Immediate, binäre Addition mit Konstante als Quelle
ADDQ	Add Quick, addiere schnell (Zahlenwert <= 8)
ADDX	Add Extend, addiere mit Extendbit
AND	Logical And, logisches Und
ANDI	And Immeadiate, logisches Und mit einer Konstanten
ASL	Arithmetik Shift Left, arithmetisches Linksschieben
ASR	Arithmetik Shift Right, arithmetisches Rechtsschieben
Bcc	Branch conditionally, bedingter Sprung (s. Bedingungen)
BRA	Branch Always, bedingungsloses Springen
BCC	Branch Carry Clear, Sprung bei Carry = 0
BCS	Branch Carry Set, Sprung bei Carry = 1
BEQ	Branch Equal, Sprung bei Gleichheit
BGE	Branch Geater or Equal, Sprung bei größer oder gleich
BGT	Branch Geater Than, Sprung bei größer als
BHI	Branch Higher, Sprung bei höher
BLE	Branch Less or Equal, Sprung bei kleiner oder gleich
BLS	Branch Lower or Same, Sprung bei ... oder ...
BLT	Branch Less Then, Sprung bei kleiner als
BMI	Branch Minus, Sprung bei minus
BNE	Branch Not Equal, Sprung bei ungleich
BPL	Branch Plus, Sprung bei plus
BVC	Branch Overflow Clear, Sprung Überlauf = 0
BVS	Branch Overflow Set, Sprung Überlauf = 1
BCHG	Bit Test and Change, Bit testen und verändern
BCLR	Bit Clear, Bit löschen
BRA	Branch Always, unbedingter Sprung
BSET	Bit Set, Bit setzen
BSR	Branch to Subroutine, Sprung zum Unterprogramm
BTST	Bit Test, Bit testen
CHK	Check Register against Boundaries, Register prüfen
CLR	Clear Operand, Operand löschen
CMP	Compare, vergleiche
CMPA	s. CMP, andere Adressierungsart
CMPI	s. CMP, andere Adressierungsart
CMPM	s. CMP, andere Adressierungsart

DBcc	Test Condition, Decrement and Branch, prüfe Bedingung, dekrementiere und Springe
DIVS	Divide Signed, vorzeichenbehaftete Division
DIVU	Divide Unsigned, vorzeichenlose Division
EQR	Exclusive OR, exklusives Oder
EORI	Exclusive OR Immediate, exklusives Oder mit Konstanten
EXG	Exchange Registers, vertausche Register
EXT	Sign Extendes
ILLLEGAL	Illegal, löst einen Illegal-Interrupt (Exception) aus
JMP	Jump, unbedingter Sprung
JSR	Jump to Subroutine, Sprung zum Unterprogramm
LEA	Load Effective Address, lade die effektive Adresse
LINK	Link Stack, verbinde den Stack
LSL	Logical Shift Left, logisches Linksschieben
LSR	Logical Shift Right, logisches Rechtsschieben
MOVE	Move, verschiebe
MOVEA	Move Address, verschiebe Adressen
MOVEM	Move Multiple Registers, verschiebe mehrere Register
MOVEP	Move Peripheral Data, verschiebe Daten zur Peripherie
MOVEQ	Move Quick, verschiebe schnell (nur 8-Bit-Konstante)
MULS	Multiply Signed, mulipliziere vorzeichenrichtig
MULU	Multiply Unsigned, multipliziere ohne Vorzeichen
NBCD	Negate Decimal with Extend, BCD-Zahl negieren
NEG	Negate Binary, negiere Binär
NEGX	Negate Extend, negiere mit Berücksichtigung des Extendbit
NOP	No Operation, keine Operation (4 Taktzyklen lang)
NOT	NOT (One's Complement), nicht (komplementieren)
OR	Logical Or, logisches Oder
ORI	Logical Or Immediate, logisches Oder mit Konstanter
PEA	Push Effektive Address, push effektive Adresse
RESET	Reset External Devices, Rücksetzen externer Geräte
ROL	Rotate Left, nach links rotieren
ROR	Rotate Right, nach rechts rotieren
ROXL	Rotate Left with Extend, mit Erweiterung nach links rotieren
ROXR	Rotate Right with Extend, mit Erweiterung nach rechts rotieren
RTE	Return from Exception, Rückkehr von einem Interrupt
RTR	Return and Restore, Rückkehr und Wiederherstellen
RTS	Return from Subroutine, Rückkehr aus einem Unterprogramm
SBCD	Subtract Decimal with Extend, Subtrahiere Dezimal mit Erweiterung
Scc	Set Conditionally, setze wenn Bedingung erfüllt
STOP	Stop, anhalten
SUB	Subtract Binary, subtrahiere Binär

SUBA	Subtract Address, subtrahiere binär mit Addressregister als Ziel
SUBI	Subtract Immediate, subtrahiere binär mit Konstanten als Quelle
SUBQ	Subtract Quick, subtrahiere schnell
SUBX	Subtract Extended, subtrahiere mit Extendbit
SWAP	Swap Register Halves, vertausche Registerhälften
TAS	Test and Set Operand, teste und setze Operand
TRAP	Trap, Falle (Anspringen der TRAP-Verktoren)
TRAPV	Trap on Overflow, Anspringen von Trap #7 bei Überlauf
TST	Test, testen
UNLK	Unlink, Gegenteil zu Link

Tabelle der Sprungbedingungen

In den Befehlen mit einem cc kann eine der folgenden Buchstabenkombinationen eingegeben werden

RA	Branch Always, bedingungsloses Springen
CC	Branch Carry Clear, Sprung bei Carry = 0
CS	Branch Carry Set, Sprung bei Carry = 1
EQ	Branch Equal, Sprung bei Gleichheit
GE	Branch Geater or Equal, Sprung bei größer oder gleich
GT	Branch Geater Than, Sprung bei größer als
HI	Branch Higher, Sprung bei höher
LE	Branch Less or Equal, Sprung bei kleiner oder gleich
LS	Branch Lower or Same, Sprung bei niedriger oder identisch
LT	Branch Less Then, Sprung bei kleiner als
MI	Branch Minus, Sprung bei minus
NE	Branch Not Equal, Sprung bei ungleich
PL	Branch Plus, Sprung bei plus
VC	Branch Overflow Clear, Sprung Überlauf = 0
VS	Branch Overflow Set, Sprung Überlauf = 1

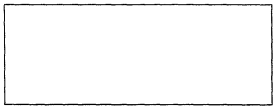
9.4 ASCII-Tabelle

Der Zeichensatz des Personalcomputers umfaßt 255 Zeichen, wobei die ersten 127 mit dem internationalen 7-Bit-ASCII-Code identisch sind. Die nächsten 127 Zeichen beinhalten die nationalen europäischen Sonderzeichen, häufig verwendete mathematische Zeichen sowie Zeichen einer Pseudografik.

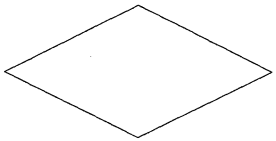
Dec	Hex	Zeichen		Dec	Hex	Zeichen	Dec	Hex	Zeichen	Dec	Hex	Zeichen
0	00	^@	NUL	32	20	SPC	64	40	@	96	60	'
1	01	^A	SOH	33	21	!	65	41	A	97	61	a
2	02	^B	STX	34	22	"	66	42	B	98	62	b
3	03	^C	ETX	35	23	#	67	43	C	99	63	c
4	04	^D	EOT	36	24	$	68	44	D	100	64	d
5	05	^E	ENQ	37	25	%	69	45	E	101	65	e
6	06	^F	ACK	38	26	&	70	46	F	102	66	f
7	07	^G	BEL	39	27	'	71	47	G	103	67	g
8	08	^H	BS	40	28	(	72	48	H	104	68	h
9	09	^I	HT	41	29	)	73	49	I	105	69	i
10	0A	^J	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	^K	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	^L	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	^M	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	^N	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	^O	SI	47	2F	/	79	4F	O	111	6F	o
16	10	^P	DEL	48	30	0	80	50	P	112	70	p
17	11	^Q	DC1	49	31	1	81	51	Q	113	71	q
18	12	^R	DC2	50	32	2	82	52	R	114	72	r
19	13	^S	DC3	51	33	3	83	53	S	115	73	s
20	14	^T	DC4	52	34	4	84	54	T	116	74	t
21	15	^U	NAK	53	35	5	85	55	U	117	75	u
22	16	^V	SYN	54	36	6	86	56	V	118	76	v
23	17	^W	ETB	55	37	7	87	57	W	119	77	w
24	18	^X	CAN	56	38	8	88	58	X	120	78	x
25	19	^Y	EM	57	39	9	89	59	Y	121	79	Y
26	1A	^Z	SUB	58	3A	:	90	5A	Z	122	7A	Z
27	1B	^[	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	^\	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	^]	GD	61	3D	=	93	5D	]	125	7D	}
30	1E	^^	RS	62	3E	<	94	5E	^	126	7E	~
31	1F	^_	US	63	3F	?	95	5F	_	127	7F	DEL

9.5 Symbole zur Erstellung von Flußdiagrammen

Die Grundlage für die Symbole zum Erstellen von Flußdigrammen ist die DIN 66001 »Sinnbilder für Datenfluß- und Programmablaufpläne«:

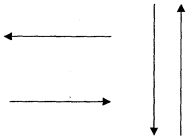


Allgemeine Operation
z.B. Daten transferieren, logische und arithmetische Operationen u.ä.

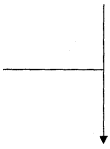


Verzweigung
z.B. bei Entscheidungen und Vergleichen

Grenzstelle
z.B. mit der Beschriftung Beginn und Ende für den Programmanfang und das Programmende



Ablauflinien zeigen den Programmablauf an



Zusammenführung
z.B. nach verschiedenen Programmteilen nach Entscheidungen

Unterprogramm

9.6 Bezugsquelle für weitere Interface-Karten

Weitere Interface-Karten sind bei folgender Adresse erhältlich:

Firma Richard Schmidt
Zülpicher Str. 240
5000 Köln 41

Die Firma Richard Schmidt liefert sowohl Prototypenkarten für den Atari ST wie für IBM-kompatible PCs. Bei Anfragen und Bestellungen bitte den Rechnertyp angeben.

Die Firma Richard Schmidt ist auf die Entwicklung von Hard- und Softwarelösungen im Bereich Messen, Steuern, Regeln spezialisiert: Maschinensteuerungen, Meß- und Betriebsdatenerfassungen und -auswertungen, Laborsoftware. Unsere Hauptkunden kommen aus dem Bereich des Maschinenbaus. Aber auch Firmen, die für bestehende oder neue Anlagen eine Betriebsdatenerfassung oder Steuerung brauchen, zählen zu unseren Kunden.

In Zusammenarbeit mit verschiedenen Firmen des Maschinenbaus entwickeln und bauen wir auch Prüfstände, z.B. Ultraschallprüfung, Widerstandsmessung, Bildverarbeitung, Schrittmotorprüfung, Weg- und Dickenmessung, Kraft-Weg-Messung und vieles mehr.

Wir versuchen, unsere Projekte möglichst mit käuflichen Standardkomponenten zu realisieren, und benutzen z.B. die Protoypenkarte nur in Ausnahmefällen oder auf Kundenwunsch.

Auch bei der Software können wir auf einen erheblichen Bestand an Modulen zurückgreifen, die wir für neue Projekte verwenden und dem Kunden nur noch mit einer Lizenzgebühr berechnen.

9.7 Literaturverzeichnis

Analog Devices: **Data-Acquisition Databook 1982**
Analog Devices Norwood, Massachusetts 02062, USA.

Arbeitskreis der Dozenten der Regelungstechnik:
Regelungstechnik in der Versorgungstechnik
Verlag C.F. Müller GmbH, Karlsruhe, 1983, ISBN 3-7880-7318-7

Beuth: **DIN-Taschenbuch 166, Informationsverarbeitung (1–4)**
DIN-Verlag, 1981, ISBN 3-410-11382-7

K. Beuth, O. Beuth: **Elementare Elektronik**
Vogel-Verlag, 1987, ISBN 3-8023-0828-X

W. Böhm: **Elektronisch steuern**
Vogel-Verlag, 1986, ISBN 3-8023-0806-9

H.P. Blomeyer-Bartenstein: **Datenkommunikation und lokale Computer-Netzwerke**
Markt&Technik-Verlag, 1983/1985, ISBN 3-89090-376-2

A. Bode, W. Händer: **Rechnerarchitektur**
(Band 1: Grundlagen und Verfahren); (Band 2: Strukturen)
Springer-Verlag, 1980/1983, ISBN 3-540-09656-6 und ISBN 3-540-12267-2

Brückmann, Englisch, Gerits: **ST Intern**
Data Becker GmbH, 1987, ISBN 3-89011-119-X

Bürgel: **Neue Normen und Schaltzeichen der digitalen Informationsverarbeitung**
Franzis-Verlag, 1978, ISBN 3-7723-6491-8

Dittrich: **Der Data Becker Führer Atari ST**
Data Becker GmbH, 1987, ISBN 3-89011-418-0

F. G. Duncan: **Microprocessor Programming and Software Development**
Prentice/Hall, 1979, ISBN 0-13-581405-7

T. Ebel: **Regelungstechnik**
Teubner Studienskripten, 1987, ISBN 3-519-40057-X

T. Ebel: **Beispiele und Aufgaben zur Regelungtechnik**
Teubner Studienskripten, 1981, ISBN 3-519-10070-3

Elektronik IVC: **Mikrocomputer-Lehrbuch**
Pflaum Verlag München, 1983, 3. Auflage, ISBN 3-7905-0383-5

Georg Färber: **Bussysteme**
R.Oldenburg Verlag, 1984, ISBN 3-486-28581-5

H. Geschwinde: **Einführung in die PLL-Technik**
Vieweg, 1978, ISBN 3-528-04077-7

Günter Friedrichs, Adam Schaff: **Auf Gedeih und Verderb**
Europa Verlag, 1982, ISBN 3-203-50799-4

Herter, Röcker, Lörcher: **Nachrichtentechnik**
Hanser Verlag, 1981, ISBN 3-446-12907-3

Hilf, Nausch: **M68000-Familie** (Band 1 und Band 2)
tewi-Verlag, 1984, ISBN 3-921803-16-0

Hofer: **Datenfernverarbeitung**
Springer-Verlag, 1978, ISBN 3-540-08621-8

J. Hübner: **Moderne Analog-Schaltungen**
Markt&Technik-Verlag, 1986, ISBN 3-89090-153-0

Intel: **Microprocessor and Peripheral Handbook**
Intel Corporation, Santa Clara, 1982

Intel: **Memory Components Handbook**
Intel Corporation, Santa Clara, 1982

W. Heinlein: **Grundlagen der faseroptischen Übertragungstechnik**
Teubner Studienbücher, 1985, ISBN 3-519-06117-1

Jankowski, Reschke, Rabich: **Atari ST Profibuch**
Sybex-Verlag, 1988, ISBN 3-88745-563-0

Reiner Klar: **Digitale Rechenautomaten**
Walter de Gruyter-Verlag, 1983, ISBN 3-11-008372-8

Werner Kobitzsch: **Mikroprozessoren, Aufbau und Wirkungsweise**
Teil 1 und Teil 2
Oldenbourg Verlag, 1979/1980, ISBN 3-486-20801-2 und ISBN 3-486-21611-2

Wen C. Lin: **Handbook of Digital Systems Design for Scientists and Engineers**
CRC Press, Inc., 1981, ISBN 0-8493-0670-1

Wen C. Lin: **Microprocessors: Fundamentals & Applications**
IEEE Press, New York, 1977

Link: **Assembler-Programmierung**
Franzis-Verlag, 1988, ISBN 3-7723-8831-0

Löffelmann, Plege: **Atari ST GFA-Basic**
Markt&Technik-Verlag, 1988, ISBN 3-89090-553-6

R. Mäusl, E.Schlagheck: **Meßverfahren in der Nachrichtentechnik**
Hünthig-Verlag, 1986, ISBN 3-7785-0922-5

P. McCorduck: **Denkmaschinen**
Markt&Technik-Verlag, 1987, ISBN 3-89090-419-X

MMI: Bipolar **LSI Data Book**
Monolithic Memories Incorporated, 1980

G. Möllmann: **Atari ST Programmierhandbuch**
Markt&Technik-Verlag, ISBN 3-89090-524-2

P. Schicker: **Datenübertragung und Rechnernetze**
B.G.-Teubner-Verlag, ISBN 3-519-02463-2

P. Schicker: **Datenübertragung und Rechnernetze**
B.G.-Teubner-Verlag, Stuttgart, 1983, ISBN 3-519-02463-2

V. Schmidt: **Digitalschaltungen mit Mikroprozessoren**
B.G.-Teubner-Verlag, Stuttgart, 1981, ISBN 3-519-02452-7

Schröder, Rommel: **Elektrische Nachrichtentechnik**
Hüthig & Pflaum-Verlag, 1978, ISBN 3-8101-0045-5

Siemens: **Mikrocomputer Bausteine**
Band 1, Band 2, Band 3 (ab 1979)
Datenbuch Siemens, Siemens AG, Balanstr. 73, 8000 München

Texas Instruments: **Einführung in die Mikroprozessortechnik**
Texas Instruments GmbH Deutschland, 1977, ISBN 3-88078-009-9

Klaus-Dieter Thies: **Die 8085/8086 Interfaces**
tewi-Verlag, 1983, ISBN 3-921-803-23-3

Valvo: **Digitaltechnik mit integrierten Schaltungen**
Valvo GmbH Hamburg 1 (ab 1971)

H. Wallner, P. Hilgers: **Mikroprozessoren**
BI-Wissenschaftsverlag, 1980, ISBN 3-411-01569-1

H. Weber: **Laplace-Transformationen für Ingenieure der Elektrotechnik**,
Teubner Studienskripten, ISBN 3-519-40069-3

Wollschlaeger: **Atari ST Assembler-Buch**
Markt&Technik-Verlag, 1987, ISBN 3-89090-467-X

P. Wratil, R. Schmidt: **PC/XT/AT-Messen, Steuern, Regeln**
Markt&Technik-Verlag, 1987, ISBN 3-89090-477-7

Zaks, Lesea: **Mikroprozessor Interface Techniken**
Sybex Verlag, Düsseldorf, 1983, ISBN 3-88745-012-4

Zirpel: **Operationsverstärker**
Franzis Elektronik-Fachbuch, 1986, ISBN 3-7723-6134-X

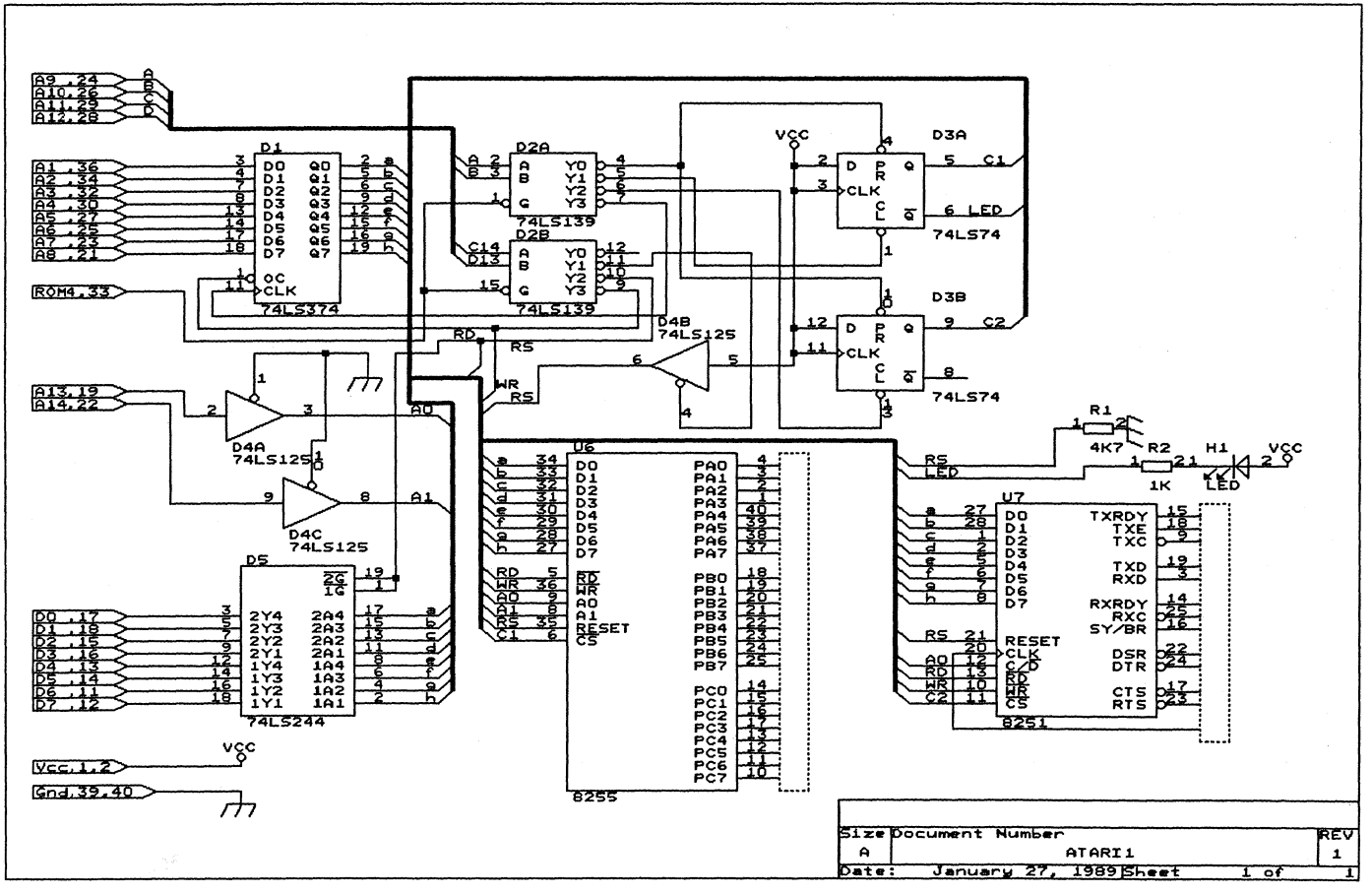


Bild 8.6: Die Schaltung der Interface-Platine (Logikteil)

Stichwortverzeichnis

1772 87
2149 90
4-Bit-Prozessor 38
4004 38
68000 105
6850 89
68901 80
7-Schichten-Modell 61
8008 38
8080 38
8251 6, 98
8255 91
82C55 91

A
Abakus 37
absolute Adressierung 157
Abtast-Halte-Glied 138
ACKNLG-Signal 65
ADC 217
Addierer 128
Addition 44
Adreßbus 53
Adressierung, absolute 157
– Konstanten- 157
–, unmittelbare 157
Adressierungsart 156
Adreßbus 42
Adreßregister indirekt 158
– indirekt mit Adreßdistanz 160
– indirekt mit Adreßdistanz und Index 161
– indirekt mit Postdekrement 160
– indirekt mit Postinkrement 159
ALGOL 37
Algorithmus 39
ALU 44
Analog-Wandlung 130
Analog/Digital-Wandler 121
Analog/Digital-Wandlung 133
Analogrechner 37
Anlaufschritt 67
Anpassung 38
Anstiegsbegrenzer 129
Applikations-Schicht 61
Arbeitssicherheit 37
ARI 158
Arithmetical and Logical Unit 44
arithmetischer Befehl 44, 166
ARI 158
ASCII-Norm 69
Assembler 145, 163 f., 190
asynchrones Verfahren 69
Atari-ROM-Port 178
ATN 59
Ausgangssignal, sinusförmig 212
–, trapezförmig 212

B
Babbage 37
Bardeen 38
Basisstrom 215
Baud-Rate 67 f., 242
Baudot-Code 70
Bauform 173
Bauteile 173
Befehlscode 163
Befehlszeiger 152
Bestückungsseite 174
Betriebspannung 209
Bit Set/Reset 94
Bit 13
BLANK 108

Blockschaltbild 93
Brattain 38
Burks 38
Bussystem 51f.
Bustreiber 62
BUSY-Leitung 65
Bus 42

C
CAS 108
CCITT-Empfehlung 70
Celsius-Skala 224
Central Processing Unit 41
Centronics 65
Centronics-Norm 204
Chip 37
Clock-Rate 45
CMOS-IC 177
Codefehler 142
Command Instruction 100
Commander 58
Compiler 146
Control Unit 44
Controller 73
COUNT8B 185
CP/M 38
CPU 12, 41
Custom Chip 105

D
D-Regler 232
Darstellung, zeitsynchron 194
Datei, relokative 190
Datenbus 42, 53
Datensicherung 71
Datentransferbefehle 165
Datentransferrate 62
Datentypen 151
Datenverbindung 51
DAV-Signal 59
DCA 207
DE 108
Dekoder 181
Dienste 60
Differential-Regler 232
Differentialschnittstellen 71
differentielle Nichtlinearität 141
Differenzeingangsspannung 125
Differenzverstärker 122
Digital/Analog-Wandler 132
direkte Registeradressierung 157
Dividierer 128
Division 44
DMA 109, 113
DMA-Controller 113
Doppelrampenverfahren 135
DREIECK 210
Dreiecksfunktion 210
Drifterscheinungen 128
DRUCK 205
Druckerinterface 204
DTACK 114
Dual Slope 136
dynamischer Speicher 103

E
E/A-Bereiche 54
EBCDI-Code 70
ECB-Bus 56
Editor 190
EEPROM 103
Eigenschwingungen 125
Einbitfehler 101
Eingabe, getriggert 196
Eingangsspannung 215
Electric EPROM 103
Elektrometer 127
Emitterstrom 215
EPROM 103
Erasable PROM 103
Ereignistrigger 194
Ersatztypen 173
EXECUTE-Zyklus 43
Exponential-Funktion 224
External 192

F
Fehler der Konversionsverfahren 141
Fehlererkennungsverfahren 71
Feldeffekttransistor 216
Festwertspeicher 103
FETCH-Zyklus 43

Flag-Register 44
FLASH 190
Flash-ADC 138
Flußdiagramm 39, 148
Frequenz/Spannungs-Wandlung 132
Full Duplex 67, 98
Funktionen 192
Funktionsgenerator 210

G
Gegenkopplung 126
Gesamtverstärkung 126
Gleitkommaarithmetik 37
Glue 108
Goldstine 38
Grundfrequenz 241

H
Half Duplex 67
Heizeffekt 234
HEX-Files 72
HP-IP 58
HSYNC 108
Hysterese 125, 240

I
I-Regler 232
I/O-Controller 73
IEC 58
IEEE-488-Bussystem 58
Inline-Assembler 192
INPUT8B 187
Integral-Regler 232
integrale Nichtlinearität 143
Integrationstechnik 37
INTEL 38, 56
Interfaceadapter 183
Interpreter 146
Interruptbefehle 171
Interruptleitungen 55
Interruptstruktur 114
Interrupt-Vektoren 114
invertierender Eingang 123
–, Verstärker 126
ISO 61
ISO-Modell 255
Istwert 232

K
Kellerspeicher 154
Kelvin-Skala 224
Kollektorstrom 216
Komparatorschaltung 124
Komplementbildung 44
Konstanten-Adressierung 158
Kontrollbus 53
Konversionlinie 143
Konversionsverfahren 138
Konversionszeit 133
Kreuzsicherungsverfahren 72
Kurvenform 210

L
LAN 253
LAN-Verbindung 255
Langzeitinstabilität 135
Laufzeitdiagramm 64
LDS 105
Least Significant Bit 130
LEDBLK 191
Leistungstransistor 216
Linker 190
Listener 58
Logikanalysator 194
LOGIK_A 197
logische Befehle 166
logischer Befehl 44
lokales Netz 52, 71, 251
Lötseite 174
Lötstopplack 174
Lötzinn 174

M
Makrodefinition 191
Maschinensprache 145
MASTST 192
Mikroprogrammschritt 45
Mikroprogramm 45
Mitkopplung 125
Mittelwertbilder 129

MMU 108
Mode Definition 94
–, Instruction 100
Modem 99
Monotoniefehler 142
motorische Vorgänge 231
MOTOROLA-Serie 56
MOVE 151
MULTIBUS 56
Multiplikation 44
Multiplizierer 128, 140

N
Negative Temperature Coefficient 223
Netzwerk-Schicht 61
nichtinvertierender Eingang 123
NMI 114
Normungen 69
NRFD 59
NTC-Thermometer 224
NTC-Widerstand 223

O
Offsetfehler 143
OPA 207
OPCODE-FETCH 41
Opcode 163
Open-Collector 64
Operand 163
Operation Code 163
Operationsverstärker 122
Oszillator 100

P
Parallel-ADC 138
parallele Bussysteme 53
parallele Verbindung 65
paralleler Transportmechanismus 52
parasitäre Kapazität 62
Paritätsbit 71, 101
Parität 101
Pascal 37, 190
PC-relative Adressierung 162
–, Adressierung mit Index 162
Speicherbausteine 6
Peripherie 43
PE 65
Physikalische Schicht 61
PL/1 37
Planartransistor 38
Präsentations-Schicht 61
problemorientiert 146
Program Counter 150
Programm 39
Programm-File 190
Programmable ROM 103
Programmierer 39
Programmiersprachen 145
Programmunterbrechung 55
PROM 103
Proportionalregeler 232
Prozeduren 192
Pseudo-Schreib-Funktion 181
Punkt-zu-Punkt-Verbindung 52

Q
Quantisierung, digitale 214
Quantisierungseinheit 141
Quantisierungsfehler 141
Quantisierungsprünge 141
Quantisierungsschritt 133
Quarz 240
Quellregister 45
Quersumme 71

R
R/2R-Netzwerk 130
RAMPE 211
Rampen-ADC 221
Rampenverfahren 133, 217
RAM 103
Random Access 103
RAS 108
Read Only Memory 103
Receive Data 99
Receiver Clock 99
–, Ready 99
Redundanz 71
Referenzspannung 209
Regelverhalten 232
Registeradressierung, direkte 157
Registerstruktur 150

Register 41
Regler 232
Ringteilnehmer 255
Ringverbindung 51
ROM-Port 182
ROM 103
RS 232 C_Terminal 242
RS 422 71
RS 485 71
Rückkopplungswiderstand 125
R_ADC 217

S
S100-Bus 57
Sample and Hold 138
Schiebebefehle 44
Schockely 38
Schreib-/Lesespeicher 103
Sensor 121, 223
Sequence Unit 44
serielle Ausgabe 247
–, Dateneingabe 244
serielle Verbindung 67
serieller Transportmechanismus 52
serielles Interface 240
SER_IN 245
SER_OUT 248
Sessions-Schicht 61
Shifter 109
SHOW-B 193
Signalausgabe, analoge 209
Single Slope 133
Sollwert 232
Source 190
Spannungs/Frequenz-Wandlung 140
Spannungsfolger 127
Spannungsquelle 212
Spannungsrampe 215
Speicherbausteine 103
Speicheroszilloskop 226
Speicherzugriffen 41
Sperrschritt 67
Sprungbefehl 41, 169
Stack 165
Stapelspeicher 154
Start-Bit 67
State 45
statischer Speicher 103
Status Register 150
Stellglied 240
Stellgröße 232
sternförmige Verbindung 51
Stop-Bit 67
Störgrößen 232
STROBE 189
Strobe-Funktion 189
Strobe-Leitung 189
STROBE-Signal 65
Stromflußrichtung 215
stromgetriebene Schnittstelle 71
Stromquelle 215
Stromspiegel 215
Struktur von Zentralprozessoren 38
Subtrahierer 128
Subtraktion 44
sukzessive Approximation 136, 217, 221
Supervisor Mode 152
–, Stack Pointer 150
Synchronbetrieb 69
Synchronisation 189
S_ADC 221
S_COPE 226

T
Takt-Signal 241
Takt 45
Talker 58
Taylor 224
Temperatur/Zeit-Diagramm 233
Temperaturdifferenz 131
Temperaturmessung 223
Temperaturstabilisierung 231
TERMINAL 250
Totzeit 232
Transistor 38
Transmit Data 99
Transportleitung 52
Transportmedien 53
Transportschicht 61
Transportverfahren 51

Tri-State-Zustand 63
Tri-State 63
TTL-IC 177

U
Übernahmesignal 189
UDS 105
unmittelbare Adressierung 158
Unterprogrammtechnik 170
User Mode 152
–, Stack Pointer 150

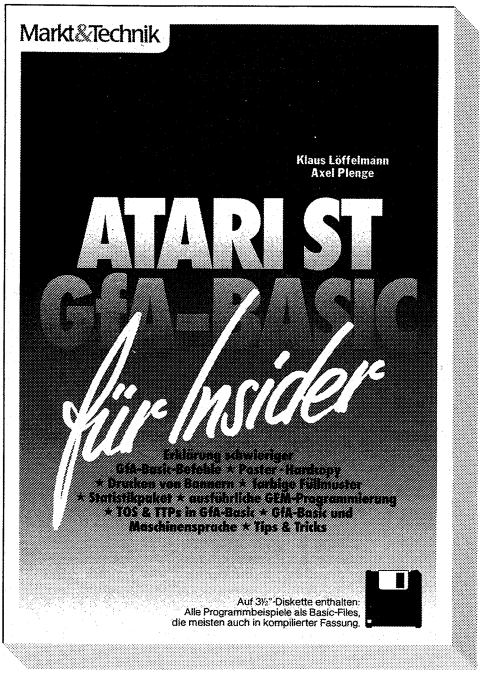
V
V-24 242
Verbindungsschicht 61
Verstärkung 126
VMEbus 56
von Neumann 37f.
VPA 108, 114
VSYNC 108

W
Wägeverfahren 137
Wärmekapazität 240
Wasserstandsregler 231
Wichtungsverfahren 221
Widerstandsmessung 215
Worst-Case 64

Z
Z80 38
Zehnerdiode 123
Zentraleinheit 38, 41
Zielspeicher 45
ZILOG 38
ZN 426 209
ZPREGLER 236
Zugriffsverfahren 51
Zuse 37
Zweipunktregler 231, 233

Bücher zum Atari ST

J. Muuss/W. Besenthal
Atari ST Programmierpraxis GFA-Basic 3.0
Dieses Handbuch zur Programmierung mit dem neuen GFA-Basic 3.0 enthält alle Informationen, die Sie zum Schreiben professioneller Programme benötigen. Nach einer eingehenden Erläuterung der Besonderheiten des GFA-Basic 3.0 werden folgende Schwerpunkte behandelt: GEM-Programmierung, Interrupt-Programmierung, Betriebssystem-Programmierung, Dateiverwaltung, 3-D-Grafik, Farbgrafik, Ray-Tracing und vieles mehr. Die Erklärung der Themen wird durch eine Vielzahl von Programmbeispielen ergänzt.
1989, 324 Seiten,
inkl. Diskette
ISBN 3-89090-702-4
DM 59,–
(sFr 54,30/öS 460,–)



K. Löffelmann/A. Plenge
Atari ST GFA-Basic für Insider
Posterhardcopy, Drucken von Bannern, farbige Füllmuster, Statistikpaket, ausführliche GEM-Programmierung, TOS & TTPs in GFA-Basic und Maschinensprache, Tips & Tricks.
1988, 218 Seiten,
inkl. Diskette
ISBN 3-89090-553-6
DM 59,–
(sFr 54,30/öS 460,–)

P. Wollschlaeger
Atari ST Programmierpraxis ST Pascal Plus
Das Handbuch für den ST Pascal Plus setzt bei soliden Pascal-Kenntnissen an: Nach einer grundlegenden Einführung in ST Pascal Plus und die Programmiersprache Pascal wird anhand vieler Beispiele gezeigt, wie unter Pascal maschinennah programmiert, Line-A-Grafiken entworfen und GEM aufgerufen wird. Das Kapitel »Multitasking und Desk Accessories« vermittelt Hintergrundwissen mit Beispielen zu Druckereinstellung, Drucker-Spooler, Hardcopy und Uhr. Abgerundet wird das Ganze mit einer Fülle von Informationen zu Datenstrukturen und Algorithmen.
1987, 261 Seiten,
inkl. Diskette
ISBN 3-89090-490-4
DM 59,–
(sFr 54,30/öS 460,–)

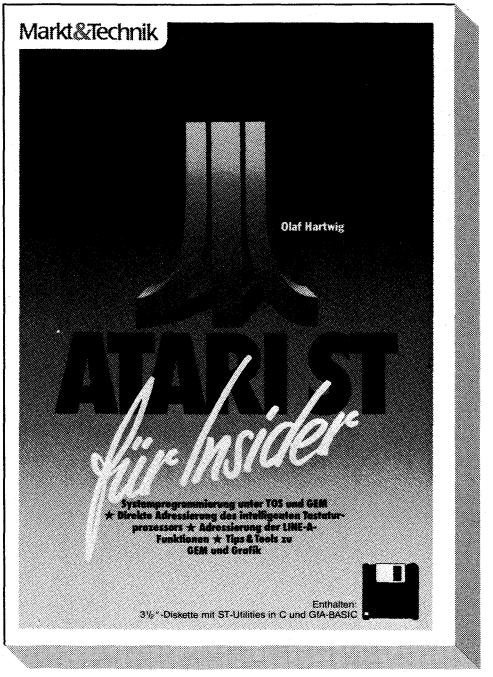
Markt&Technik
Zeitschriften · Bücher
Software · Schulung

418/906

Markt&Technik-Produkte erhalten Sie bei Ihrem Buch- oder Computerfachhändler

Bücher zum Atari ST

P. Wollschlaeger
Atari-ST-Assembler-Buch
Dieses Buch beweist, daß die Assembler-Programmierung ganz einfach ist! Ein gewisses Maß an Grundwissen über computerinterne Dinge wird jedoch vorausgesetzt. Nach einem Minimum an Theorie geht dieses Buch sofort in die Praxis. Assembler-Befehle und TOS-Funktionen werden anhand kleiner Programme erklärt, die ständig schwieriger werden, jedoch das erforderliche Wissen von Fall zu Fall mitliefern. Mit einem allgemein verwendbaren Rahmenprogramm, einer RAM-Disk, einem Disketten-Monitor und vielen kleinen Utilities bleibt für den Leser vieles, was er immer wieder verwenden kann. Dem Buch liegt eine 3½"-Diskette mit allen Beispielprogrammen bei.
1987, 300 Seiten, inkl. Diskette
Bestell-Nr. 90467
ISBN 3-89090-467-X
DM 59,–
(sFr 54,30/öS 460,-)



O. Hartwig
Atari ST für Insider
Dieses Buch eröffnet allen engagierten Atari-ST-Anwendern eine Fülle von neuen Möglichkeiten. Im Mittelpunkt stehen Tips&Tricks zu TOS und zu GEM.

Hard- und Software-Anforderungen:
Atari-ST-Computer (260 ST, 520 ST oder 1040 ST), GFA- oder ST Basic oder C-Compiler.
1987, 299 Seiten, inkl. Diskette
Bestell-Nr. 90423
ISBN 3-89090-423-8
DM 49,–
(sFr 45,10/öS 382,-)

R. Aumiller/D. Luda/G. Möllmann
Atari-ST-GEM-Programmierung in C
Dieses Buch zeigt Ihnen, wie Sie die Vorteile von GEM mit Hilfe der Programmiersprache C noch besser nutzen können. Sie werden bald in der Lage sein, Ihre eigenen GEM-Programme zu erstellen.
In zwei Teile gegliedert, einen AES- und einen VDI-Teil, werden alle Funktionen ausführlich erklärt und anhand von praxisbezogenen Beispielen erläutert. Die kompletten Programme sind im Listing-Teil des Buches zu finden sowie auf der beiliegenden 3½"-Diskette. Um alle Möglichkeiten dieses Buches voll ausschöpfen zu können, sollten Sie Grundkenntnisse in C mitbringen.
1987, 639 Seiten, inkl. Diskette
Bestell-Nr. 90488
ISBN 3-89090-488-2
DM 69,–
(sFr 63,50/öS 538,-)

Markt&Technik
Zeitschriften · Bücher
Software · Schulung

Markt&Technik-Produkte erhalten Sie bei Ihrem Buchhändler, in Computer-Fachgeschäften oder in den Fachabteilungen der Warenhäuser.

416/901

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0

Atari ST Messen Steuern Regeln

Die Autoren:

RICHARD SCHMIDT, geboren 1955 in Stettin, aufgewachsen in Moers, studierte Physik in Köln. Von 1983 bis 1988 arbeitete er als wissenschaftlicher Mitarbeiter am I. Physikalischen Institut in Köln. Danach gründete er die Firma Schmidt und Beier, deren Betätigungsfeld in der Software-Entwicklung im Bereich der Meß-, Steuer- und Regelungstechnik liegt.

Dr. PETER WRATIL, geboren 1951. Während seines Physikstudiums beschäftigte er sich vor allem mit der Meßdatenverarbeitung auf Rechnersystemen. Danach unterrichtete er Studenten in der Mikroprozessortechnologie. Seit Oktober 1984 ist er Entwicklungsleiter in der Elektronik bei der Firma Klöckner-Moeller in Bonn.

Seit Ende 1985 ist die Atari-ST-Reihe auf dem Markt, und seitdem verkaufen sich diese preisgünstigen Computer blendend – ein Beweis für die gute Konzeption und das breite Anwendungsspektrum, das sich diesem Computer eröffnet hat.
Das Hard- und Software-Angebot ist dem Markt angepaßt, aber wie steht es mit Erweiterungen am ROM-Port, dem einzigen Steckplatz der ST-Computer? Hier wurde bis heute nicht sehr viel angeboten. Deshalb entwickelten die Autoren eine Interface-Karte für den Atari ST, die getestet und unbestückt dem Buch beiliegt. Sie bestücken den festen Logikteil mit handelsüblichen Bauteilen und können anschließend auf dem Anwenderteil eigene Lösungen frei konfigurieren. Interessante Beispiele für so eine kostengünstige Erweiterung finden Sie ausführlich beschrieben in diesem Buch.
Alle dazu notwendigen Kenntnisse werden Ihnen vermittelt. Zuerst erfahren Sie in einem theoretischen Teil alles Wissenswerte über die Logikentwicklung, die Struktur und Funktionweise des 68000-Prozessors und die verschiedenen Möglichkeiten des Datentransports. Aus welchen Baugruppen und Einzelkomponenten der Atari ST aufgebaut ist und wie diese im System zusammenarbeiten, ist der Inhalt der weiteren Kapitel. Dabei wird immer ein besonderes Augenmerk auf die Interface-Technik gelegt. Wie Sie die Karte mittels GFA-Basic, ProPascal und Assembler ansprechen, wird im folgenden Kapitel erklärt.
Kapitel 8, das sich intensiv mit den Anwendungen der Interface-Karte beschäftigt, ist wohl der interessanteste Teil des Buches. Logikanalysator (mit Steuerprogramm), regelbare Strom- und Spannungsquelle, Anschluß von Sensoren, Speicheroszilloskop (mit Steuerprogramm) und serielle bzw. parallele Datenübertragung sind einige der Schaltungen, die im Buch genau beschrieben werden und deren Nachbau ohne Schwierigkeiten möglich ist.
Mit den gewonnenen Kenntnissen sind Sie in der Lage, individuelle Anwendungen selbst zu realisieren. Damit richtet sich das Buch nicht nur an den Elektronikbastler, sondern auch an den professionelle Hardware-Entwickler.

Die Begleitdiskette:
Auf ihr sind alle Beispielprogramme in GFA-Basic 3.0 bzw. ProPascal 1.1 im Quellcode enthalten.

Software-Anforderungen:
GFA-Basic 3.0

Hardware-Anforderungen:
Atari ST oder Mega ST

ISB N 3-89090-679-6

Markt&Technik

4 001057 906792

DM 98,-
sFr 90,20
öS 764,-